IN THE UNITED STATES DISTRICT COURT
FOR THE SOUTHERN DISTRICT OF NEW YORK

INTERNATIONAL BUSINESS MACHINES CORPORATION,

Plaintiff,

-vs.-

PLATFORM SOLUTIONS, INC. and T3 TECHNOLOGIES, INC.,

Defendants.

Civil Action No. 06 CV 13565 (LAK)

SUPPLEMENTAL DECLARATION OF EDWARD J. DEFRANCO IN SUPPORT OF PLAINTIFF IBM'S REPLY CLAIM CONSTRUCTION BRIEF FOR THE *MARKMAN* HEARING

I, Edward J. DeFranco, under penalty of perjury, declare as follows:

I am a partner at Quinn Emanuel Urquhart Oliver & Hedges, LLP, and I represent Plaintiff International Business Machines Corporation in this case. I submit this Declaration in Support of Plaintiff's Reply Claim Construction Brief for the *Markman* Hearing.

1. Attached to this declaration as Exhibit 47 is a true and correct copy of page 409 of the 10th Edition of the IBM Dictionary of Computing, © 1994.

2. Attached to this declaration as Exhibit 48 is a true and correct copy of U.S. Patent No. 5,446,860.

3. Attached to this declaration as Exhibit 49 is a true and correct copy of U.S. Patent No. 5,416,916.

4. Attached to this declaration as Exhibit 50 is a true and correct copy of U.S. Patent No. 6,112,274.

5. Attached to this declaration as Exhibit 51 is a true and correct copy of U.S. Patent No. 6,223,196.

6. Attached to this Declaration as Exhibit 52 is a true and correct copy of a webpage from PSI's website, available at http://www.platform-solutions.com/aboutPSI.php, accessed on June 26, 2008.

7. Attached to this Declaration as Exhibit 53 is a true and correct copy of a letter from PSI's counsel to IBM's counsel, which was received on June 25, 2008.

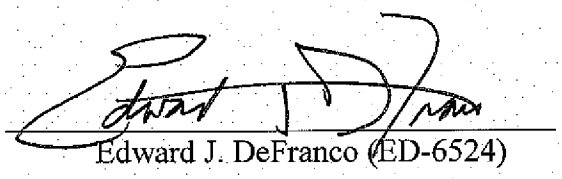
8. Attached to this Declaration as Exhibit 54 is an updated chart of the remaining patent claim terms/phrases in dispute, along with the parties' proposed constructions.

9. Attached to this Declaration as Exhibit 55 is a true and correct copy of "Requirements, Bottlenecks, and Good Fortune: Agent for Microprocessor Evolution," Yale Patt, Proceedings of the IEEE, Vol. 89, No. 11, November 2001.

10. Attached to this Declaration as Exhibit 56 is a true and correct copy of the 4/11/97 Notice of Allowability sent by the United States Patent and Trademark Office in connection with the patent application that resulted in U.S. Pat. No. 5,687,106.

I swear under penalty of perjury that the foregoing is true and correct.

Dated: June 27, 2008



Edward J. DeFranco (ED-6524)

IEEE

Dictionary of Computing

- The most comprehensive computing dictionary ever published
- More than 18,000 entries

Limitation of Liability

While the Editor and Publisher of this book have made reasonable efforts to ensure the accuracy and timeliness of the information contained herein, neither the Editor nor the Publisher shall have any liability with respect to loss or damage caused or alleged to be caused by reliance on any information contained herein.

4 5 6 7 8 9 0 DOC/DOC 9 9 8 7 6

ISBN 0-07-031488-8 (HC)
ISBN 0-07-031489-6 (PBK)

The sponsoring editor for this book was Daniel A. Gonneau and the production supervisor was Thomas G. Kowalczyk.

Printed and bound by R. R. Donnelley & Sons Company.

Tenth Edition (August 1993)

This is a major revision of the *IBM Dictionary of Computing,* SC20-1699-8, which is made obsolete by this edition. Changes are made periodically to the information provided herein.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country. Comments may be addressed to IBM Corporation, Department E37/656, P. O. Box 12195, Research Triangle Park, NC 27709.

International Edition

Copyright © 1994 by International Business Machines Corporation. Exclusive rights by McGraw-Hill, Inc. for manufacture and export. This book cannot be re-exported from the country to which it is consigned by McGraw-Hill. The International Edition is not available in North America.

When ordering this title, use ISBN 0-07-113383-6.

This book is printed on acid-free paper.

machine-readable information (MRI) All textual information contained in a program, such as a system control program, an application program, or microcode. MRI includes all information that is presented to or received from a user interacting with a system. This includes menus, prompts, messages, report headings, commands, and responses. MRI may appear on printers or on display panels.

machine-readable medium A medium that can convey data to a sensing device. (A) Synonymous with automated data medium.

machine-readable passport A passport that is intended to be read and verified by a machine in accordance with an ISO standard. (T) (A)

machine run The execution of one or more routines that are linked to form one operating unit.

machine-sensible information Synonym for soft copy.

machine space point Synonym for space pointer machine object.

machine storage pool In the AS/400 system and System/38, a storage pool used by the machine and certain highly shared programs.

machine vision In robotics, use of video cameras to obtain visual images that are converted to analog electrical signals and then to digital or gray-scale image data for processing. Two-dimensional and three-dimensional machine vision is used for applications such as robot guidance and automated inspection and quality control.

machine word Synonym for computer word.

MACLIB library A library that contains macros, copy files, or source program statements for use under CMS.

MAC protocol See medium access control protocol.

macro (1) In photography, a magnifying camera lens that can focus down to a few inches. (2) See macrodefinition, macro prototype statement. (3) Synonym for macroinstruction.

macro assembler A program that converts and assembles macroinstructions into machine code.

macrobending In an optical fiber, optical attenuation caused by macroscopic deviations of the axis from a straight line. (E) Contrast with microbending.

macrobend loss In an optical fiber, that loss attributable to macrobending. (E) Synonymous with curvature loss. Contrast with microbend loss.

macro call (1) A statement, embedded in a source language, that is to be replaced by a defined statement sequence in the same source language. The macro call will also specify the actual parameters for any formal parameters in the macrodefinition. (T) (2) Synonym for macroinstruction.

macrodeclaration Synonym for macrodefinition.

macrodefinition (1) A possibly parameterized specification for a statement sequence to replace a macro call. A macrodefinition may be considered as a procedure to be executed by a macrogenerator yielding the statement sequence. (T) (2) A set of statements defining the name of, format of, and conditions for generating a sequence of assembler statements from a single source statement. (3) See also library macrodefinition, source macrodefinition.

macroexpansion (1) The sequence of statements that result from a macrogeneration operation. (2) Synonym for macrogeneration.

macrogenerating program Synonym for macrogenerator.

macrogeneration An operation in which an assembler produces a sequence of assembler language statements by processing a macrodefinition called by a macroinstruction. Macrogeneration takes place at preassembly time. Synonymous with macroexpansion.

macrogenerator A program for replacing macro calls with defined statement sequences according to macrodefinitions. A macrogenerator may be an independent computer program or it may be integrated as a subprogram in a compiler or assembler to generate the source program. (T) Synonymous with macroprocessor, macrogenerating program.

macroinstruction (1) An instruction in a source language that is to be replaced by a defined sequence of instructions in the same source language and that may also specify values for parameters in the replaced instructions. Synonymous with macro, macro statement. (T) (2) In assembler programming, an assembler language statement that causes the assembler to process a predefined set of statements called a macro definition. The statements normally produced from the macro definition replace the macroinstruction in the program. See also definition statement. (3) Synonymous with macro call, macrostatement.

macrolanguage The representations and rules for writing macroinstructions and macrodefinitions.

US005446860A

United States Patent [19]

Dresser et al.

[11] **Patent Number: 5,446,860**

[45] **Date of Patent: Aug. 29, 1995**

[54] **APPARATUS FOR DETERMINING A COMPUTER MEMORY CONFIGURATION OF MEMORY MODULES USING PRESENCE DETECT BITS SHIFTED SERIALLY INTO A CONFIGURATION REGISTER**

[75] Inventors: **Scott A. Dresser**, Mountain View, Calif.; **Scott A. Markinson**, Goffstown, N.H.; **Richard B. Goud**, Concord, Mass.

[73] Assignee: **Hewlett-Packard Company**, Palo Alto, Calif.

[21] Appl. No.: **3,194**

[22] Filed: **Jan. 11, 1993**

[51] **Int. Cl.**[6] **G06F 13/00**

[52] **U.S. Cl.** **395/427**; 364/DIG. 1; 364/244.2; 364/244.5; 364/243; 364/DIG. 2; 364/246.7; 364/969.1; 364/245

[58] **Field of Search** ..... 395/425; 364/200 MS File, 400 MS File, 245, 246.7, 969.1, 244.2, 244.5, 243

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,958,222	5/1976	Messina et al.	395/425
4,234,934	11/1980	Thorsrud	395/400
4,414,627	11/1983	Nakamura	395/400
4,562,532	12/1985	Nishizawa et al.	395/425
4,571,676	2/1986	Mantellina et al.	395/425
4,654,787	3/1987	Finnell et al.	395/425
4,794,559	12/1988	Greenburger	365/49
4,882,700	11/1989	Mauritz et al.	365/51
5,040,153	8/1991	Fung et al.	365/230.03
5,179,686	1/1993	White	395/425
5,269,010	12/1993	MacDonald	395/425
5,270,964	12/1993	Bechtolsteim et al.	365/52

FOREIGN PATENT DOCUMENTS

0440445 8/1991 European Pat. Off. .

OTHER PUBLICATIONS

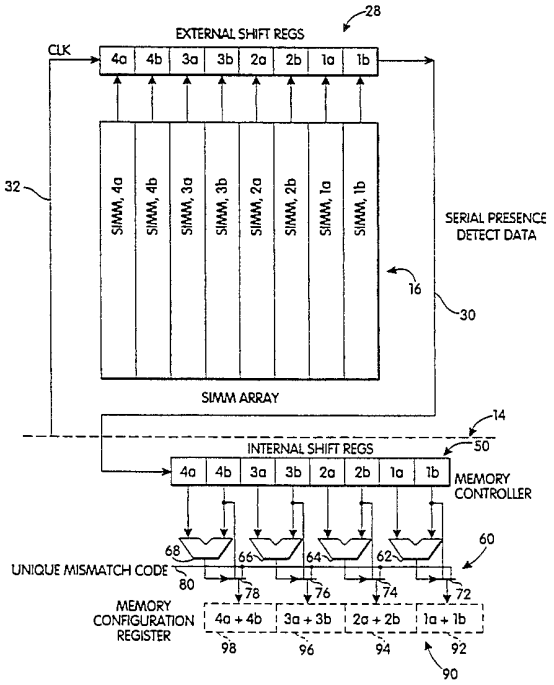
"Memory Presence Detect", IBM Technical Disclosure Bulletin, vol. 29, No. 3, Aug. 1986, p. 1381.

Primary Examiner—Glenn Gossage
Assistant Examiner—Frank J. Asta

[57] **ABSTRACT**

An apparatus is provided for determining a configuration of memory modules. Each of the memory modules produces presence detect bits. In a first feature, the apparatus includes an external register for receiving the presence detect bits from the memory modules in parallel and a memory controller integrated circuit. The memory controller integrated circuit includes an internal register for storing the presence detect bits and logic circuitry for determining a memory configuration. The presence detect bits are serially transferred from the external register to the internal register. In a second feature, the logic circuitry for determining a memory configuration includes comparator circuitry for comparing the presence detect bits of each pair of memory modules and generating a match signal or a mismatch signal. When a match signal occurs, the presence detect bits of the memory module pair are loaded into a memory configuration register. When a mismatch signal occurs, a mismatch code is loaded into the memory configuration register.

16 Claims, 4 Drawing Sheets



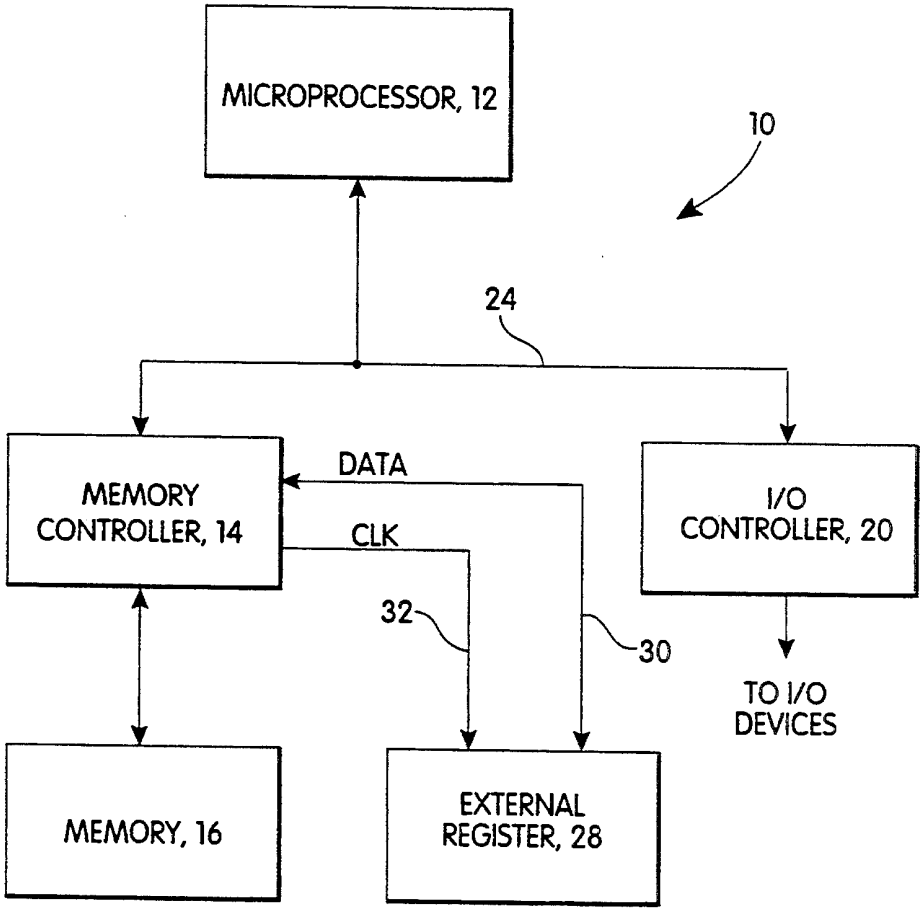

MICROPROCESSOR, 12
10
24
MEMORY CONTROLLER, 14
DATA
CLK
I/O CONTROLLER, 20
32
30
TO I/O DEVICES
MEMORY, 16
EXTERNAL REGISTER, 28

Fig. 1

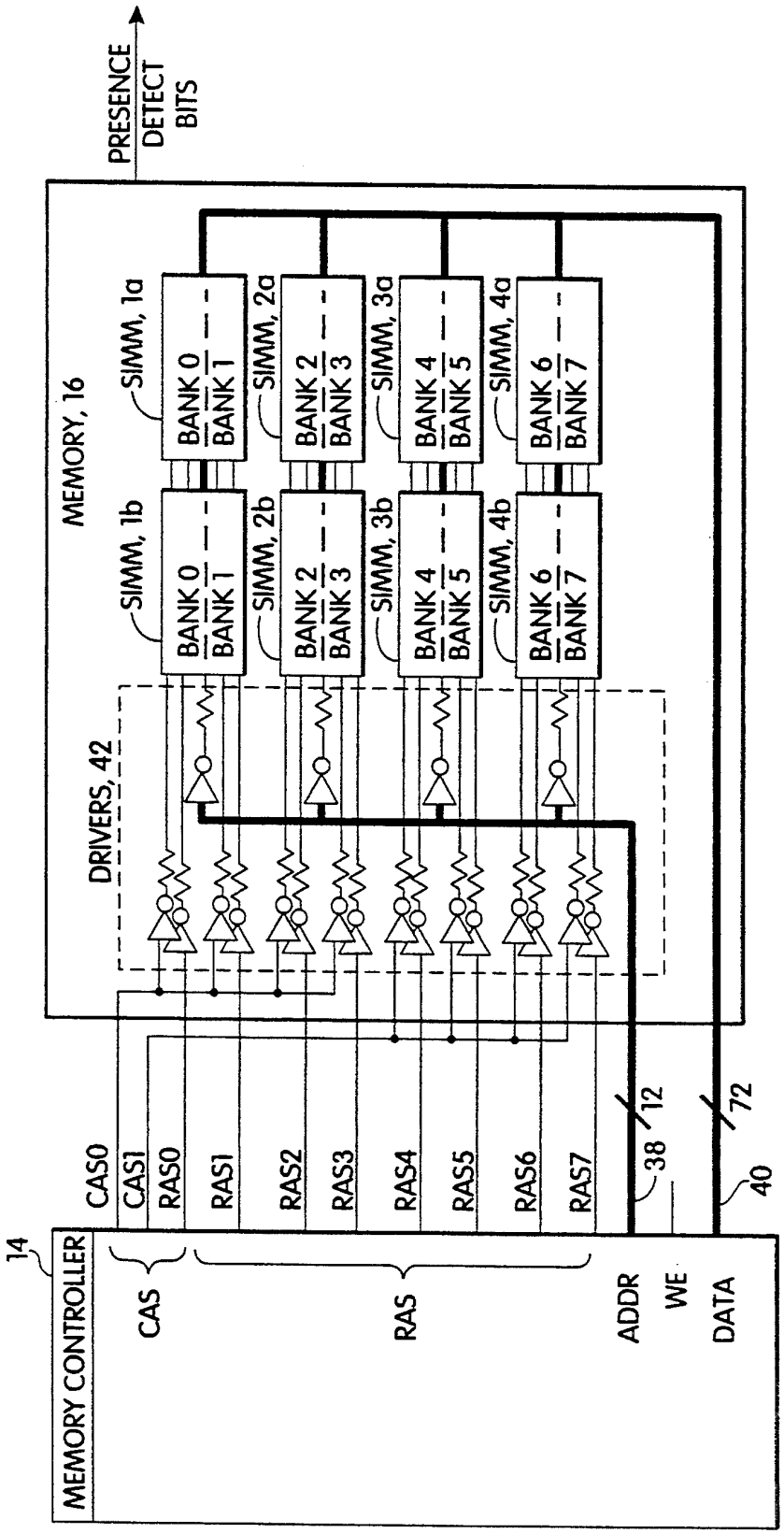

PRESENCE DETECT BITS
MEMORY, 16
SIMM, 1a
SIMM, 2a
SIMM, 3a
SIMM, 4a
SIMM, 1b
SIMM, 2b
SIMM, 3b
SIMM, 4b
BANK 0/BANK 1
BANK 2/BANK 3
BANK 4/BANK 5
BANK 6/BANK 7
DRIVERS, 42
CAS0
CAS1
RAS0
RAS1
RAS2
RAS3
RAS4
RAS5
RAS6
RAS7
12
72
38
40
14
MEMORY CONTROLLER
CAS
RAS
ADDR
WE
DATA

Fig. 2

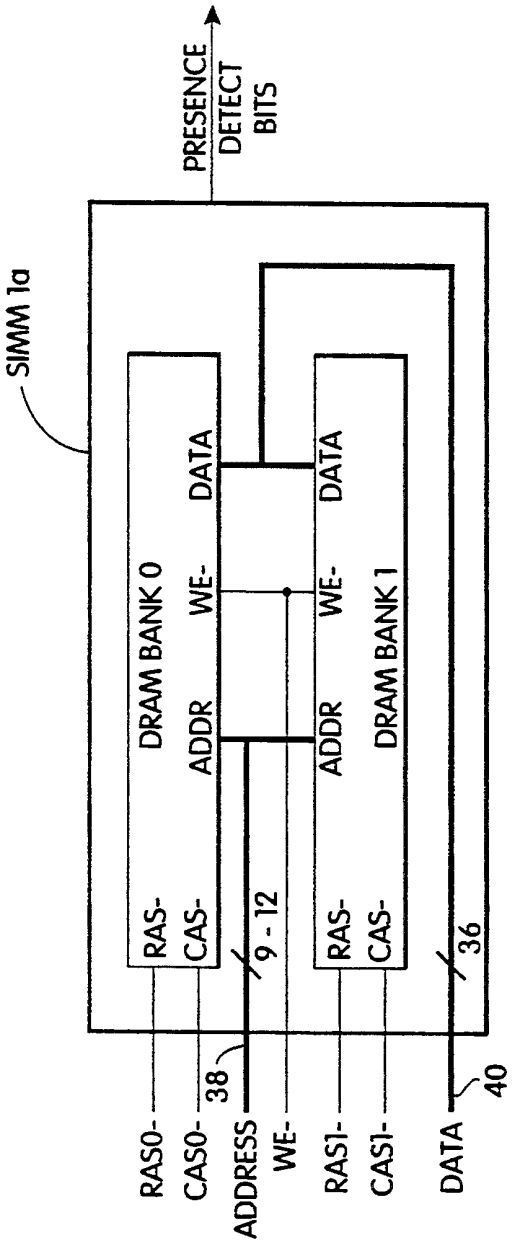

PRESENCE DETECT BITS
SIMM 1a
DRAM BANK 0
DRAM BANK 1
RAS-
CAS-
ADDR
WE-
DATA
9 - 12
36
38
40
RAS0-
CAS0-
ADDRESS
WE-
RAS1-
CAS1-
DATA

Fig. 3

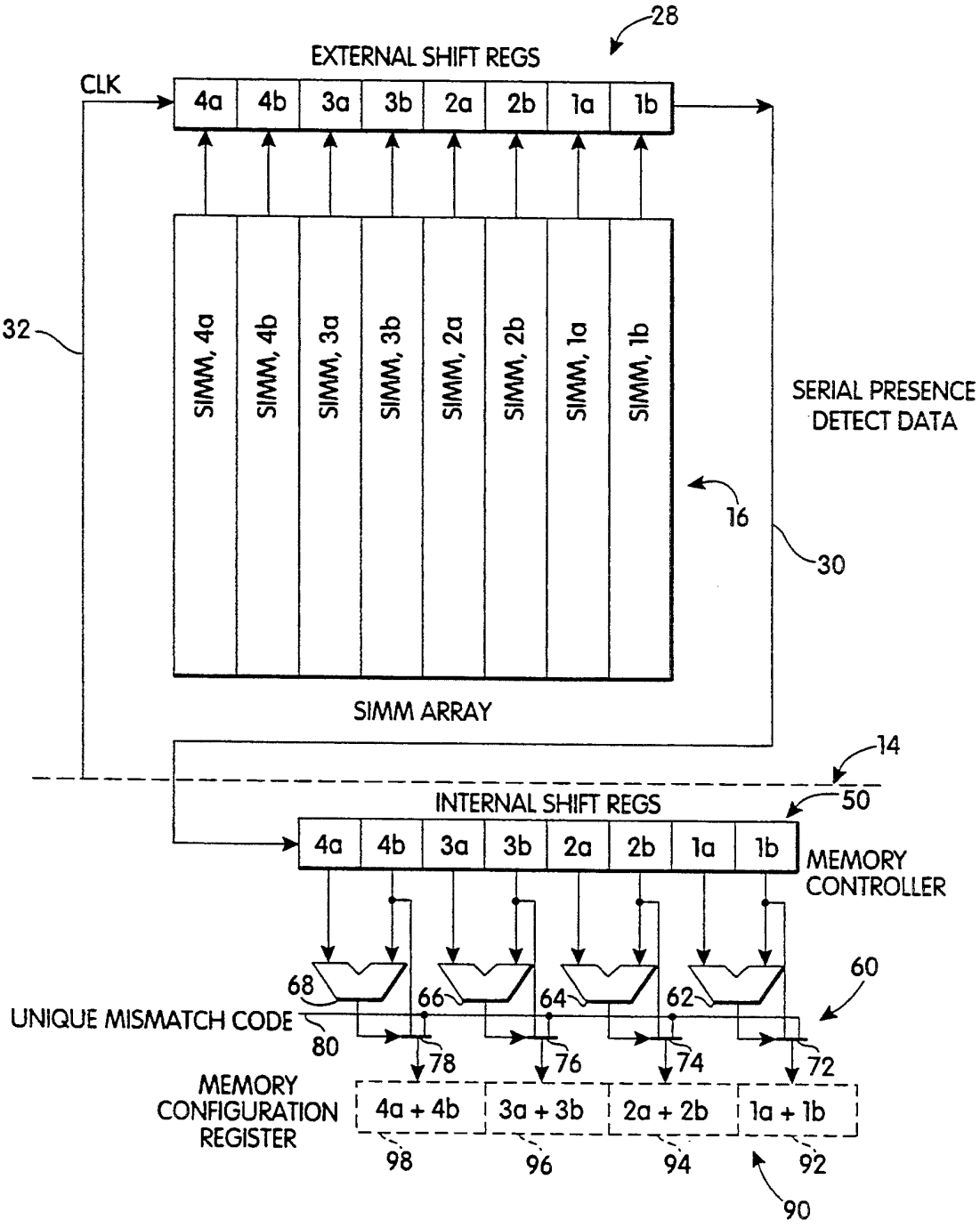

28
EXTERNAL SHIFT REGS
CLK
4a
4b
3a
3b
2a
2b
1a
1b
32
SIMM, 4a
SIMM, 4b
SIMM, 3a
SIMM, 3b
SIMM, 2a
SIMM, 2b
SIMM, 1a
SIMM, 1b
SERIAL PRESENCE DETECT DATA
16
30
SIMM ARRAY
14
INTERNAL SHIFT REGS
50
4a
4b
3a
3b
2a
2b
1a
1b
MEMORY CONTROLLER
60
68
66
64
62
UNIQUE MISMATCH CODE
80
78
76
74
72
MEMORY CONFIGURATION REGISTER
4a + 4b
3a + 3b
2a + 2b
1a + 1b
98
96
94
92
90

Fig. 4

APPARATUS FOR DETERMINING A COMPUTER MEMORY CONFIGURATION OF MEMORY MODULES USING PRESENCE DETECT BITS SHIFTED SERIALLY INTO A CONFIGURATION REGISTER

FIELD OF THE INVENTION

This invention relates to techniques for controlling computer memories and, more particularly, to an apparatus for automatically determining the configuration of a user-configurable computer memory.

BACKGROUND OF THE INVENTION

The requirements of a computer memory vary widely as to capacity, speed, data bus width, and the like, depending on the application of the computer. It has been customary for manufacturers of general purpose computers to provide modular memory subsystems. The memory subsystem includes a number of slots, or connectors, for memory modules. Memory modules may be mounted in some or all of the slots, depending on the application. The memory modules may vary as to capacity, operating speed, data bus width, etc.

A typical computer utilizes a memory controller for converting a memory address supplied by the CPU into the required address and control signals for accessing a particular memory location. The typical memory controller generates row address strobe, column address strobe and write enable signals. The memory controller is typically implemented in one or more large scale integrated circuits. When the memory is modular, the memory controller and the CPU must receive information which defines the memory configuration in order to supply the required address and control signals to the memory modules that are present in the system. It is thus customary that each memory module provides a number of presence detect codes, or presence detect bits, at its connector. The presence detect bits include information as to the memory module capacity, DRAM device speed, etc. A memory configuration is determined from the presence detect bits. An example of a memory module that provides presence detect bits is a single in-line memory module (SIMM).

A large number of presence detect bits must be handled by the memory controller. A typical memory subsystem may include up to 8 memory modules, each of which generates 6 presence detect bits, totaling 48 presence detect bits. This information must be input to the memory controller integrated circuit, where the number of I/O pins is limited. In prior art systems, presence detect bits have typically been latched in parallel because they are available in parallel from the memory modules.

In existing memory subsystems, memory modules may be utilized in pairs to achieve a wide data bus. For example, many computer systems require a 64 bit memory data bus, whereas standard memory modules are available with a 32 bit data bus. It is thus necessary that the memory modules be installed in pairs to achieve the 64 bit data bus. In this case, the memory modules of each pair must be identical. The computer system must be capable of determining when a user inadvertently installs different memory modules as a pair. In prior art systems, determining memory module pair mismatches has been performed by the system software. This approach has the disadvantages of requiring software

intervention, which takes time, and requiring a larger number of registers in the system software in order to identify memory module mismatches.

SUMMARY OF THE INVENTION

In accordance with the present invention, apparatus is provided for determining a memory configuration of memory modules for use in controlling the memory modules. Each of the memory modules produces presence detect bits. The apparatus comprises an external register for receiving and storing the presence detect bits from each of the memory modules, and a memory controller integrated circuit for controlling the memory modules. The memory controller integrated circuit includes an internal register for storing the presence detect bits. The apparatus further comprises means for serially transferring the presence detect bits from the external register to the internal register and logic means within the memory controller integrated circuit for determining a memory configuration in response to the presence detect bits. The means for serially transferring the presence detect bits preferably includes a single data line and a single clock line on the memory controller integrated circuit. In a preferred embodiment, the memory modules, the external register and the memory controller integrated circuit are mounted on a single circuit board.

The logic means for determining a memory configuration preferably comprises a memory configuration register, comparison means for comparing the presence detect bits of memory module pairs and selector means responsive to the comparison means for loading the presence detect bits of the memory module pairs into the memory configuration register when the comparison means indicates that the presence detect bits of the memory module pairs match and for loading a mismatch code into the memory configuration register when the comparison means indicates that the presence detect bits of the memory module pairs do not match. The comparison means preferably includes a comparator for each pair of memory modules. The selector means preferably includes a data selector for each pair of memory modules.

According to another aspect of the invention, an apparatus is provided for determining a memory configuration of one or more pairs of memory modules. The memory configuration is used in controlling the memory modules. Each of the memory modules produces presence detect bits. The apparatus comprises a register for receiving presence detect bits produced by each of the memory modules and for storing the presence detect bits, a memory configuration register, comparison means for comparing the presence detect bits of each pair of memory modules and generating a match signal or a mismatch signal, and selector means responsive to the match signal for loading the presence detect bits of the corresponding pair of memory modules into the memory configuration register and responsive to the mismatch signal for loading a mismatch code into the memory configuration register. The register, the memory configuration register, the comparison means and the selector means are preferably located within a memory controller integrated circuit.

BRIEF DESCRIPTION OF THE DRAWINGS

For a better understanding of the present invention, reference is made to the accompanying drawings,

which are incorporated herein by reference and in which:

FIG. **1** is a block diagram of a computer system that incorporates the present invention;

FIG. **2** is a block diagram of the memory subsystem shown in FIG. **1**;

FIG. **3** is a block diagram of a memory module used in the memory subsystem; and

FIG. **4** is a block diagram of apparatus for determining a memory configuration in accordance with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

A block diagram of a computer system incorporating the present invention is shown in FIG. **1**. A computer system **10** includes a central processing unit (CPU) such as a microprocessor **12**. A memory subsystem includes a memory controller **14** and a memory **16**. The memory controller **14** controls access by microprocessor **12** to memory **16**. The computer system **10** further includes an input output (I/O) controller **20** which controls communication with I/O devices (not shown). The microprocessor **12**, the memory controller **14** and the I/O controller **20** are connected to and communicate on a bus **24**.

The memory controller **14** is typically a large scale integrated circuit which receives an address from microprocessor **12** on bus **24** and provides control signals to memory **16** for reading or writing to a specified location in memory **16**. As described below, the memory **16** provides presence detect bits to an external register **28**. The presence detect bits are input to memory controller **14** on a data line **30** under control of a clock signal on a clock line **32**.

The memory subsystem is shown in more detail in FIG. **2**. An address bus **38** carries addresses from the memory controller **14** to the memory **16**, and a data bus **40** carries data between memory **16** and memory controller **14**. The memory controller **14** provides control signals to the memory **16**. In particular, column address strobe (CAS) signals CAS**0** and CAS**1**, and row address strobe (RAS) signals RAS**0**, RAS**1**, RAS**2**, RAS**3**, RAS**4**, RAS**5**, RAS**6** and RAS 7 are generated by memory controller **14**. The CAS**0** and CAS**1** signals are used to strobe column addresses into the dynamic random access memory (DRAM) chips of memory **16**. Similarly, the RAS**0**–RAS**7** signals are used to strobe row addresses into the DRAM chips of memory **16**. Before a location in memory **16** can be accessed, its column address and row address must be strobed into the DRAM chips using the CAS and RAS signals. A write enable (WE) signal is provided by the memory controller **14** to enable data to be written to the DRAM chips of memory **16**.

Memory **16** includes drivers **42** that receive and drive the CAS**0** and CAS**1** signals, the RAS**0**–RAS**7** signals and the addresses on address bus **38**. In the present example, the memory **16** includes 8 slots, or connectors, in which memory modules can be installed. Each slot may hold a single memory module. The user of the computer system **10** configures the system as required for a particular application by installing memory modules in the slots. In the example of FIG. **2**, each of the 8 slots is loaded with a single in-line memory module (SIMM). SIMMs are industry standard memory modules. In a preferred embodiment, the memory **16** operates with X36 ECC SIMMs. Slot **0** is loaded with SIMM **1***a*; slot **1** is loaded with SIMM **1***b*; slot **2** is loaded with SIMM **2***a*; slot **3** is located with SIMM **2***b*; slot **4** is loaded with SIMM **3***a*; slot **5** is loaded with SIMM **3***b*; slot **6** is loaded with SIMM **4***a*; and slot **7** is loaded with SIMM **4***b*. The SIMMs typically have a 36 bit data output. In the present example, the data width is 72 bits, and the SIMMs are addressed in pairs **1***a* and **1***b*, **2***a* and **2***b*, **3***a* and **3***b*, and **4***a* and **4***b*. The SIMMs of each pair (such as SIMMs **1***a* and **1***b*) are interconnected to receive the same RAS and CAS signals. The data bus **40** is connected to each pair of SIMMs such that bits **0** to **35** are connected to the first SIMM of the pair, and bits **36**–**72** are connected to the second SIMM in the pair.

Each SIMM may include up to 18 DRAM chips which may be 1 MBit chips, 4 MBit chips or 16 MBit chips. All the DRAM chips on a single SIMM are of the same memory capacity. Each of the paired SIMMs must include DRAM chips of the same memory capacity. Each SIMM may be fully populated or half populated. A fully populated SIMM includes 9 DRAM chips on each side (18 DRAM chips total). A half populated SIMM includes 9 DRAM chips on one side.

The DRAM chips on corresponding sides of each pair of SIMMs constitute a DRAM bank. For example, the DRAM chips on one side of SIMM **1***a* and SIMM **1***b* form DRAM bank **0**. The DRAM chips on the other side of SIMM **1***a* and SIMM **1***b* form DRAM bank **1**. Similarly, SIMMs **2***a* and **2***b* include DRAM banks **2** and **3**; SIMMs **3***a* and **3***b* include DRAM banks **4** and **5**; and SIMMs **4***a* and **4***b* include DRAM banks **6** and **7**.

The memory **16** outputs presence detect bits which are utilized as described below to determine a memory configuration. Each of the memory modules, or SIMMs, in the memory **16** produces 6 presence detect bits for a total of 48 bits. The 6 bits from each SIMM are encoded to indicate the memory module size and the speed of the DRAM chips.

Memory module SIMM **1***a* is shown in more detail in FIG. **3**. The other SIMMs in memory **16** have the same construction. SIMM **1***a* includes DRAM bank **0** and DRAM bank **1**. The data lines of DRAM banks **0** and **1** are connected to data bus **40**, and the address lines of DRAM banks **0** and **1** are coupled to the address bus **38**. Depending on the size of the DRAM chips on the SIMM **1***a*, the address bus may carry an address having 9 to 12 bits. DRAM bank **0** receives the RAS**0** signal and the CAS**0** signal. DRAM bank **1** receives the RAS**1** signal and the CAS**1** signal. The DRAM banks **0** and **1** both receive the write enable signal from memory controller **14**. The SIMM **1***a* produces 6 presence detect bits for determining a memory configuration as described below. Techniques for generation of RAS, CAS and write enable signals, and row and column addresses for accessing the DRAM chips of the memory modules are generally known to those skilled in the art. Desirable features of a preferred memory controller are disclosed in copending applications Ser. No. 07/938,901 filed Aug. 31 1992 and Ser. No. 07/938,113 filed Aug. 31, 1992, now U.S. Pat. No. 5,278,801, which are hereby incorporated by reference.

A block diagram of apparatus for determining a memory configuration in accordance with the present invention is shown in FIG. **4**. The SIMMs **1***a*, **1***b*, **2***a*, **2***b*, **3***a*, **3***b*, **4***a* and **4***b* supply presence detect bits in parallel to external shift register **28**. The external register **28** has sufficient capacity to store the presence detect bits for the maximum number of memory modules that may be

installed in the computer system. In the present example, the system can accept up to 8 SIMMs, each of which produces 6 presence detect bits. Thus, the external register **28** has a capacity of 48 bits. The external register **28** is preferably mounted on the same circuit board as the memory modules.

Further in accordance with the invention, the memory controller **14**, implemented as a large scale integrated circuit, contains an internal shift register **50**. The internal register **50** preferably has the same capacity as external register **28**. A clock signal generated by memory controller **14** is carried on the clock line **32** to external register **28**. After the presence detect bits have been loaded in parallel from the memory modules into shift register **28**, the clock signal is used to serially transfer the data from external register **28** to internal register **50** on the serial data line **30**. Thus, the 48 bits of presence detect data are transferred into internal register **50** within memory controller **14** using a single clock line **32** and a single data line **30**.

The presence detect data contained in internal register **50** is processed by logic circuitry **60** within the memory controller **14** to determine a memory configuration. In particular, the presence detect bits for pairs of memory modules are compared to determine whether the presence detect bits match. As discussed above, the memory modules are paired in order to obtain the desired memory data bus width. The memory modules of a pair are typically located adjacent to each other, but are not required to be located adjacent to each other. The memory modules of a pair must be identical. In particular, the presence detect bits of SIMMs **1***a* and **1***b* are compared by a comparator **62**; the presence detect bits of SIMMs **2***a* and **2***b* are compared by a comparator **64**; the presence detect bits of SIMMs **3***a* and **3***b* are compared by a comparator **66**; and the presence detect bits of SIMMs **4***a* and **4***b* are compared by a comparator **68**. The comparators **62**, **64**, **66** and **68** produce a match signal or a mismatch signal depending on the respective presence detect bits.

The outputs of comparators **62**, **64**, **66** and **68** are supplied to the select inputs of data selectors **72**, **74**, **76** and **78**, respectively. The presence detect bits of SIMM **1***b* are supplied to one input of data selector **72**; the presence detect bits of SIMM **2***b* are supplied to one input of data selector **74**; the presence detect bits of SIMM **3***b* are supplied to one input of data selector **76**; and the presence detect bits of SIMM **4***b* are supplied to one input of data selector **78**. The presence detect bits for either memory module of each pair can be input to the respective data selector, since this input is selected only when the presence detect bits of the memory modules of the pair match. It will be understood that each input and each output of comparators **62**, **64**, **66** and **68** and data selectors **72**, **74**, **76** and **78** is six bits wide in the present example.

A mismatch code is supplied on a line **80** to the second inputs of each of the data selectors **72**, **74**, **76** and **78**. The mismatch code can be any bit combination that does not match a combination of presence detect bits and which is recognized by the memory controller circuitry and the system software as indicative of a mismatch between the memory modules of a pair. The inputs to data selectors **72**, **74**, **76** and **78** are connected such that when the respective comparator output indicates a mismatch between the presence detect bits of a pair of memory modules, the mismatch code is output by the data selector. When the output of the respective comparator indicates a match between presence detect bits, the presence detect bits for the pair of memory modules are output by the data selector.

The outputs of data selectors **72**, **74**, **76** and **78** are loaded into a memory configuration register **90**. The memory configuration register **90** includes a section for storing memory configuration information representative of each pair of memory modules. Thus, a register section **92** contains memory configuration information for SIMMs **1***a* and **1***b*; a register section **94** contains memory configuration information for SIMMs **2***a* and **2***b*; a register section **96** contains memory configuration information for SIMMs **3***a* and **3***b*; and a register section **98** contains memory configuration information for SIMMs **4***a* and **4***b*. When the memory modules of a pair match, the register section is loaded with the presence detect bits for the pair of memory modules. When the memory modules of a pair do not match, the register section is loaded with the mismatch code. Because the presence detect bits for the memory modules of a pair must be identical for proper operation, the capacity of memory configuration register **90** is one half the capacity of internal register **50** and external register **28**. Thus, for the example where external register **28** and internal register **50** have a capacity of 48 bits each, the memory configuration register **90** has a capacity of only 24 bits. As a result, the memory configuration data which is supplied to the microprocessor **12** has a lower number of bits, thereby saving space in the address map.

In operation, the presence detect bits from all the memory modules are loaded in parallel into the external register **28** at system reset. After the memory controller **14** comes out of reset, it begins to serially clock data out of the external register **28** into the internal register **50**. As indicated above, this arrangement requires only two I/O pins on the memory controller **14** which are dedicated to obtaining the presence detect bits.

The memory configuration register **90** presents a single presence detect code representing a pair of memory modules to the system software when the presence detect bits from the memory modules of the pair are identical. When the presence detect bits do not match, the mismatch code is loaded into the register **90** and is presented to the system software. When the system software reads the register **90** and finds the mismatch code, the mismatch is signalled to the user. The disclosed technique requires only one code to be provided to the software for each pair of memory modules and also ensures that the two SIMMs of each pair are identical.

The present invention reduces the cost of the memory controller integrated circuit by reducing to two the number of I/O pins that is required for determining memory configuration. Another advantage is that the hardware detects memory module mismatches in paired memory systems. This provides a faster and more concise way of reporting memory configuration information to the system software.

While there have been shown and described what are at present considered the preferred embodiments of the present invention, it will be obvious to those skilled in the art that various changes and modifications may be made therein without departing from the scope of the invention as defined by the appended claims.

What is claimed is:

1. Apparatus for determining a memory configuration of memory modules for use in controlling said

memory modules, each of said memory modules producing presence detect bits, comprising:

an external register for receiving said presence detect bits from each of said memory modules in parallel and for storing said presence detect bits, said presence detect bits including encoded bits representative of memory module capacity and memory module speed;

a memory controller integrated circuit for controlling said memory modules, said external register being located external to said memory controller integrated circuit, said memory controller integrated circuit including an internal register for storing said presence detect bits;

means for serially transferring said presence detect bits from said external register to said internal register; and

logic means within said memory controller integrated circuit for determining a memory configuration in response to said presence detect bits.

2. Apparatus as defined in claim **1** wherein said means for serially transferring said presence detect bits includes a single data line and a single clock line on said memory controller integrated circuit.

3. Apparatus as defined in claim **1** wherein said memory modules are configured as memory module pairs and wherein said logic means comprises a memory configuration register, comparison means for comparing the presence detect bits of said memory module pairs and selector means responsive to said comparison means for loading the presence detect bits of one memory module of said memory module pairs into said memory configuration register when said comparison means indicates that the presence detect bits of said memory module pairs match and for loading a predefined mismatch code into said memory configuration register when said comparison means indicates that the presence detect bits of said memory module pairs do not match.

4. Apparatus as defined in claim **1** further including a circuit board for mounting said external register and said memory controller integrated circuit.

5. Apparatus as defined in claim **3** wherein said comparison means includes a comparator for each pair of memory modules and said selector means includes a data selector for each pair of memory modules.

6. Apparatus as defined in claim **5** wherein said memory configuration register includes means for storing said presence detect bits or said mismatch code for each pair of memory modules.

7. Apparatus for determining a memory configuration of one or more pairs of memory modules for use in controlling said memory modules, each memory module producing presence detect bits, comprising:

a register for receiving presence detect bits produced by each of said memory modules and for storing said presence detect bits, said presence detect bits including encoded bits representative of memory module capacity and memory module speed;

a memory configuration register;

comparison means for comparing the presence detect bits of each pair of memory modules and generating a match signal or a mismatch signal for each pair of memory modules, said match signal indicating a match between the presence detect bits of a corresponding pair of memory modules and said mismatch signal indicating a mismatch between the presence detect bits of the corresponding pair of memory modules; and

selector means responsive to said match signal for loading the presence detect bits of one memory module of the corresponding pair of memory modules into said memory configuration register and responsive to said mismatch signal for loading a predefined mismatch code into said memory configuration register.

8. Apparatus as defined in claim **7** wherein said comparison means includes a comparator for each pair of memory modules, said comparator generating a match signal or a mismatch signal.

9. Apparatus as defined in claim **8** wherein said selector means includes a data selector responsive to the match signal or the mismatch signal for each pair of memory modules.

10. Apparatus as defined in claim **7** further including a memory controller integrated circuit for controlling said memory modules, said register, said memory configuration register, said comparison means and said selector means being located within said memory controller integrated circuit.

11. Apparatus as defined in claim **10** further including means for serially transferring said presence detect bits into said register using a single data line and a single clock line on said memory controller integrated circuit.

12. Apparatus for determining a computer memory configuration, comprising:

a circuit board;

one or more pairs of memory modules mounted on said circuit board, each of said memory modules producing presence detect bits, said presence detect bits including encoded bits representative of memory module capacity and memory module speed;

an external register mounted on said circuit board for receiving said presence detect bits from each of said memory modules in parallel and for storing said presence detect bits;

a memory controller integrated circuit mounted on said circuit board for controlling said memory modules, said external register being located external to said memory controller integrated circuit, said memory controller integrated circuit including an internal register for storing said presence detect bits and logic means for determining a memory configuration in response to said presence detect bits, said memory modules being controlled by said memory controller integrated circuit in response to said memory configuration; and

means for serially transferring said presence detect bits from said external register to said internal register.

13. Apparatus as defined in claim **12** wherein said logic means includes a memory configuration register and, for each pair of memory modules, a comparator for comparing the presence detect bits of the memory modules of the pair and generating a match signal or a mismatch signal, said match signal indicating a match between the presence detect bits of the memory modules of the pair and said mismatch signal indicating a mismatch between the presence detect bits of the memory modules of the pair, and a selector responsive to said match signal for loading the presence detect bits of one memory module of the memory modules of the pair into said memory configuration register and responsive to said mismatch signal for loading a predefined mismatch code into said memory configuration register.

14. Apparatus as defined in claim 13 wherein said memory modules comprise single in-line memory modules.

15. Apparatus as defined in claim 14 wherein said means for serially transferring said presence detect bits includes a single data line and a single clock line on said memory controller integrated circuit.

16. Apparatus as defined in claim 12 wherein said external register and said internal register each comprise a serial shift register.

* * * * *

US005416916A

United States Patent [19]

Bayle

[11] Patent Number: 5,416,916

[45] Date of Patent: May 16, 1995

[54] **STRUCTURE FOR ENABLING DIRECT MEMORY-TO-MEMORY TRANSFER WITH A FLY-BY DMA UNIT**

[75] Inventor: **Shawn D. Bayle**, Chestnut Hill, Mass.

[73] Assignee: **NEC Electronics Incorporated**, Mountain View, Calif.

[21] Appl. No.: **12,613**

[22] Filed: **Feb. 2, 1993**

Related U.S. Application Data

[63] Continuation of Ser. No. 549,329, Jul. 6, 1990, abandoned.

[51] **Int. Cl.**[6] **G06F 13/00**

[52] **U.S. Cl.** **395/425**; 364/242.31; 364/DIG. 1

[58] **Field of Search** ..... 395/425, 800; 364/242.31, DIG. 1

[56] **References Cited**

U.S. PATENT DOCUMENTS

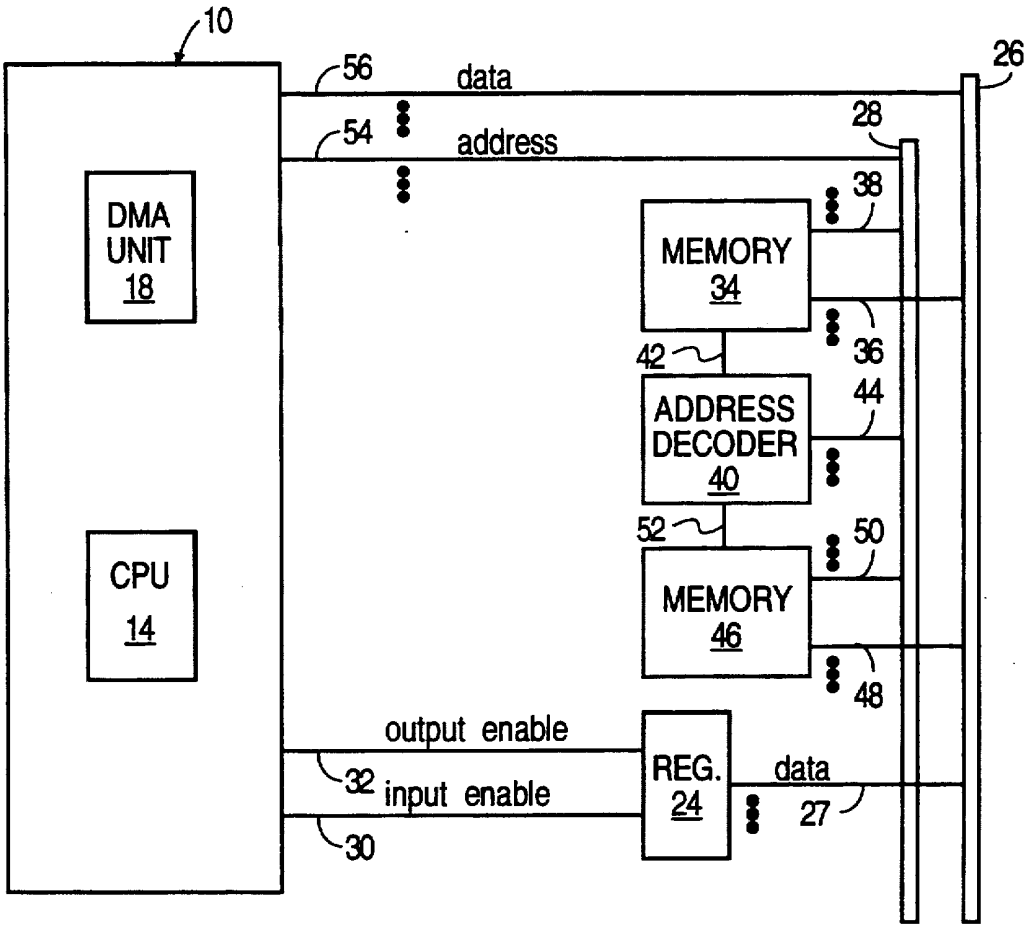
4,481,578	11/1984	Hughes et al.	395/425
5,007,012	9/1988	Dujari	395/425
5,119,485	6/1992	Ledbetter, Jr. et al.	395/425
5,193,193	3/1993	Iyer	395/725

Primary Examiner—Alyssa H. Bowler
Assistant Examiner—John Harrity
Attorney, Agent, or Firm—Skjerven, Morrill, MacPherson, Franklin & Friel; Brian D. Ogonowsky

[57] **ABSTRACT**

In a microprocessor such as a Model V40 or V50, which includes a DMA unit but does not have direct memory-to-memory transfer capabilities, an external register is used and controlled by the DMA unit to temporarily store data from memory during a direct memory-to-memory transfer by the DMA,

8 Claims, 1 Drawing Sheet



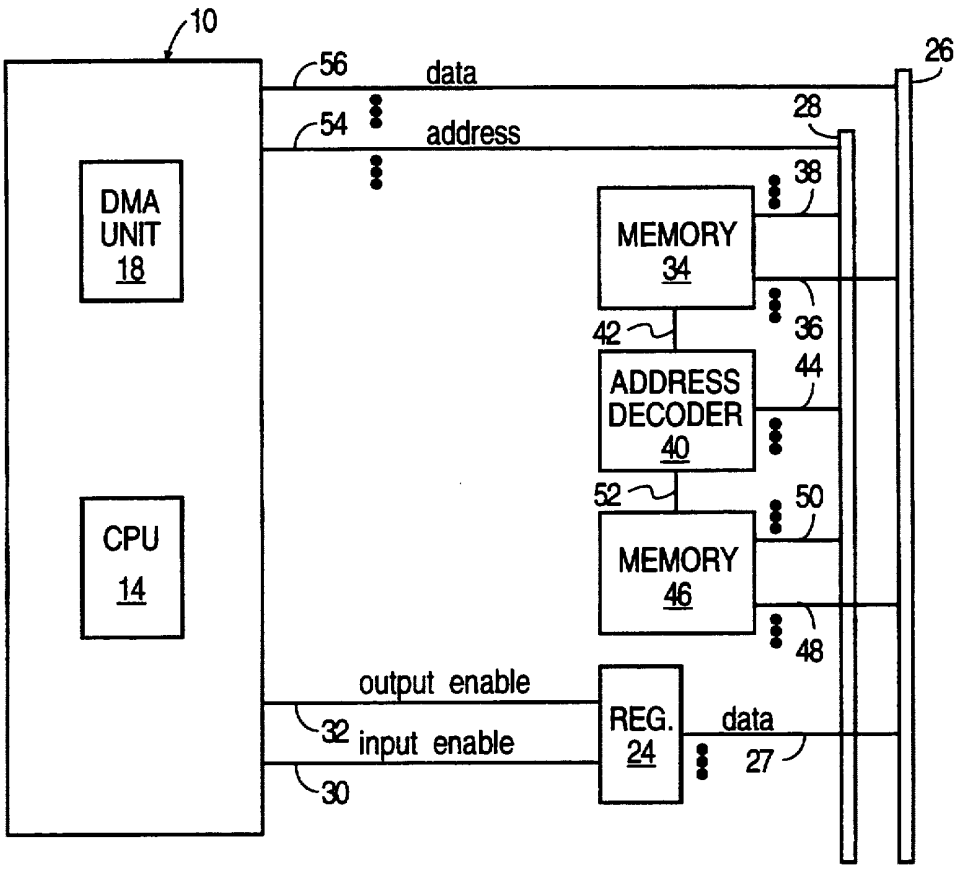

10
56
data
26
54
address
28
DMA UNIT 18
38
MEMORY 34
36
42
ADDRESS DECODER 40
44
52
50
CPU 14
MEMORY 46
48
output enable
32
input enable
REG. 24
data
27
30

FIG. 1

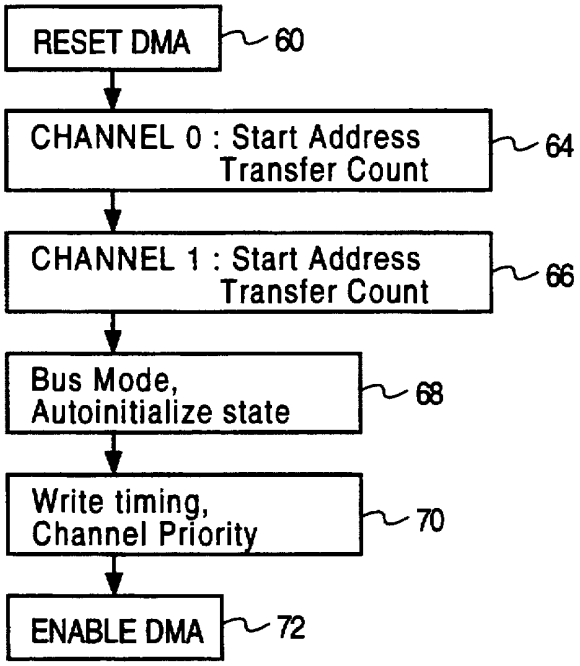

RESET DMA
60
CHANNEL 0 : Start Address Transfer Count
64
CHANNEL 1 : Start Address Transfer Count
66
Bus Mode, Autoinitialize state
68
Write timing, Channel Priority
70
ENABLE DMA
72

FIG. 2

STRUCTURE FOR ENABLING DIRECT MEMORY-TO-MEMORY TRANSFER WITH A FLY-BY DMA UNIT

This application is a continuation of application Ser. No. 07/549,329, filed Jul. 6, 1990 now abandoned.

FIELD OF THE INVENTION

This invention relates to direct memory address (DMA) controllers and in particular to a circuit for enabling a DMA controller to have a memory-to-memory transfer capability when one is not inherently provided in the DMA controller.

BACKGROUND OF THE INVENTION

Direct memory access (DMA) controllers are well known in the art. One such DMA controller is the μPD71071 manufactured by NEC Electronics Inc. Typically, a DMA controller is configured as a separated integrated circuit chip with input and output pins. The function of this discrete type of DMA controller is to provide high-speed data transfers between peripheral devices and memory or between one memory location and another memory location without the need for the central processor unit (CPU) itself to read or write any data. The DMA controller is programmed by the user with instructions identifying certain data to be transferred from one location to another. When the DMA controller is later given a simple data transfer request, the DMA controller carries out the programmed data transfer instructions. In this way, the CPU is free to perform other operations during the time when the DMA controller is transferring data from one location to another.

In a standard discrete DMA controller such as the μPD71071, a register is included within the DMA controller to temporarily store data during a memory-to-memory transfer. The data read from a source memory location must be temporarily stored in the register located inside the DMA controller until the address of the destination memory location has been provided on the address bus by the DMA controller, and the addressed memory location is ready to receive the data.

When a memory-to-input/output (I/O) device transfer is conducted, the data from the source memory may be applied directly to the input of the I/O device without temporarily storing the data, since no address bits are needed to address the selected I/O device. This is called fly-by DMA.

Additionally, data transfer from an I/O device to a memory location may also be conducted without temporarily storing the data, since once the memory location is addressed, the data from the I/O device may simply be directly read into the addressed location without the need for temporary storage by the CPU or DMA controller.

In microprocessors which include a CPU and a DMA controller, die area may be conserved by eliminating the temporary registers internal to the DMA controller which would be used for temporarily storing data during a direct memory-to-memory transfer. These microprocessors not containing temporary registers in the DMA controller, such as the Model V40 (μPD70208) or V50 (μPD70216) microprocessors by NEC Electronics Inc., are capable of providing direct memory-to-I/O device transfers or I/O device-to-memory transfers but not direct memory-to-memory transfers.

The microprocessors without this direct memory-to-memory transfer feature are generally satisfactory to a user who is willing to use some limited CPU time, and use the registers internal to the CPU, to route data between one memory location and another memory location. However, for some users, it is important that the CPU not expend even this limited CPU processing time conducting memory-to-memory transfers.

Thus, what is needed in the art is a circuit which can be used to quickly and easily enable microprocessors, such as the Model V40 and V50, to have direct memory-to-memory transfer capabilities without requiring manipulation of the data by the CPU.

SUMMARY OF THE INVENTION

For a microprocessor containing a CPU and limited DMA controller capability which provides direct memory-to-I/O device transfers and vice versa but does not provide direct memory-to-memory transfers due to the inability of the DMA controller internal to the microprocessor chip to temporarily store data read from a memory device, the following circuit may be used to enable direct memory-to-memory transfers. This inventive circuit consists of a microprocessor, such as a Model V40 or a V50, connected to an address and data bus. An external register is connected to the data bus and controlled by the microprocessor to temporarily store data during direct memory-to-memory transfers.

Thus, using this configuration, a microprocessor having a relatively small die area by eliminating temporary registers internal to the DMA controller is still available to satisfy a typical user's needs; however, when a user also desires a direct memory-to-memory transfer capability, the microprocessor may be used in conjunction with an external register to provide this direct memory-to-memory capability.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows a preferred embodiment of the invention.

FIG. **2** shows an example of an initialization flow chart for a DMA unit within a microprocessor.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. **1** shows a preferred embodiment of the invention which includes microprocessor **10**, which may be a Model V40, V50, or other microprocessor. A detailed description of the Model V40 and V50 microprocessors may be found in the data sheets for these devices obtained from NEC Electronics, Inc., incorporated herein by reference. Microprocessor **10** includes a central processing unit (CPU) **14** and a direct memory access (DMA) control unit **18**. In a preferred embodiment, CPU **14** is a Model V20 or V30 CPU manufactured by NEC Electronics, Inc. DMA control unit **18**, in a preferred embodiment, may be a modified μPD71071 DMA controller.

The Model V40 or V50 microprocessor contains a DMA control unit without a direct memory-to-memory transfer capability due to the lack of an internal register to temporarily store data while addressing the destination memory address location. Thus, the DMA unit in the Model V40 and V50 only provides direct memory-to-I/O transfers and I/O-to-memory transfers. Both of

these transfers will be referred to herein as memory-to-I/O transfers.

To provide direct memory-to-memory capability, as shown in FIG. **1**, an external 16-bit or 8-bit register **24** has data ports connected to data bus **26** via lines **27**. One type of 8-bit register which may be used is the Model 74LS374. Data bus **26** may also act as an address bus if data and address bits are multiplexed. In the embodiment of FIG. **1**, a separate address bus **28** is used. Control line **30** connects a data input enable pin of register **24** to an appropriate pin of microprocessor **10** for providing an enable signal to register **24** to enable register **24** to store the data on data bus **26**. Control line **32** connects an output enable pin of register **24** to an appropriate pin of microprocessor **10** for providing an enable signal to register **24** to enable register **24** to output the temporarily stored data onto data bus **26**.

A first memory device **34** is connected to data bus **26** via lines **36** and connected to address bus **28** via lines **38**. A chip select pin of memory device **34** is connected to an output of address decoder **40** via line **42**. Address input ports of address decoder **40** are coupled to appropriate address bus lines via lines **44** for providing chip select address signals.

A second memory device **46** is connected to data bus **26** via lines **48** and connected to address bus **28** via lines **50**. A chip select pin of memory device **46** is connected to an output of address decoder **40** via line **52**.

Microprocessor **10** has address and data ports connected to address bus **28** and data bus **26**, respectively, via lines **54** and **56**, assuming there is no multiplexing of address and data bits.

Enable and chip select signals provided by microprocessor **10** to register **24**, memory **34**, and memory **46** are generated by DMA unit **18**, as necessary, to transfer data from memory **34** to memory **46** or visa versa in accordance with a data transfer program stored in DMA unit **18**. In a typical DMA unit, separate channels within the DMA unit are used to control the transfer of data between one device and another device. For a channel within the DMA unit to control the transfer of data to or from a certain memory device, the following characteristics are typically programmed into the channel:

starting address for the transfer
transfer count
DMA operating mode
transfer size (byte/word units)

In the embodiment shown in FIG. **1**, a first channel of DMA unit **18** is programmed to control the transfer of data into or out of first memory **34**, while a second channel is programmed to control the transfer of data into or out of second memory **46**. Additional channels in the DMA unit **18** may be used for memory-to-I/O data transfers (i.e., fly-by service).

To transfer data directly from memory device **34** to memory device **46**, DMA unit **18** is requested to first directly transfer addressed data from the source memory device **34** into register **24** using a first channel of the DMA unit **18**, where register **24** is effectively treated as an I/O device. On the next cycle, a second channel of the DMA unit **18** transfers the contents of register **24** to the addressed location in the destination memory device **46** to complete the memory-to-memory transfer. Again, register **24** is effectively treated as an I/O device.

Additionally, data may be transferred from one location in memory device **34** into another location in memory device **34**, or data may be transferred from one location in memory device **46** into another location in memory device **46**, using the same technique.

Thus, using the configuration of FIG. **1**, two channels of the DMA unit **18** are used to achieve this memory-to-memory capability. In the Model V40 and V50 microprocessors, DMA unit **18** has four channels, thus leaving two channels available for fly-by service.

A microprocessor incorporating a DMA unit would normally be programmed during initialization procedures to initialize the channels within the DMA unit to perform this direct memory-to-memory transfer. FIG. **2** shows a flow diagram of a sample initialization procedure which may be easily written into software by one of ordinary skill in the art. First, in step **60**, the various registers in the DMA unit are reset. In steps **64** and **66**, the start addresses and transfer counts for the first and second channels are then loaded into the appropriate DMA unit registers. Next, in step **68**, the DMA operating modes are selected. In the case of FIG. **2**, which assumes the use of a Model V40 or V50 microprocessor, the bus mode and auto-initialize state are selected. Finally, in steps **70** and **72**, the write timing and channel priority are then set followed by the enablement of the DMA unit.

During operation of microprocessor **10**, while DMA unit **18** controls the transfer of data, whether memory-to-memory or memory-to-I/O, pursuant to a request to do so, CPU **14** may perform other operations not requiring data bus **26** or address bus **28**. The DMA unit and the CPU communicate to determine which is to control the address bus and data bus.

DMA unit **18** within microprocessor **10**, when connected as shown in FIG. **1**, is particularly useful when transferring blocks of data from a hard disk drive to the main memory of a system so that the CPU may act on the new data in the main memory. Additionally, the DMA unit **18** is frequently used for extended memory management, where blocks of data are transferred from external RAM to the main memory.

Thus, a novel circuit and method for performing direct memory-to-memory transfers has been described.

The above-described invention may be applied to various types of microprocessors and is not limited to the Model V40 or V50 microprocessor. Additionally, although a DMA unit has been shown in FIG. **1** as being connected within a micro-processor, the DMA unit may be a separate unit or may be incorporated in an integrated circuit along with other types of circuitry.

While particular embodiments of the present invention have been shown and described, it will be obvious to those skilled in the art that changes and modifications may be made without departing from this invention in its broader aspects and, therefore, the appended claims are to encompass within their scope all such changes and modifications as fall within the true spirit and scope of this invention.

What is claimed is:

1. A circuit connected to a direct memory access unit formed as an integrated circuit, wherein said direct memory access unit does not have, internal to said integrated circuit, a temporary storage register for temporarily storing data during a memory-to-memory transfer, said circuit comprising:

a bus directly connected to said integrated circuit;

two or more memory devices connected to said bus in parallel with said integrated circuit; and

an external register, external to said integrated circuit and connected to said bus in parallel with said

memory devices, said external register having one or more control terminals coupled to receive control signals generated by said integrated circuit, said external register being controlled by said integrated circuit to temporarily store data only during a direct memory-to-memory transfer operation while one of said memory devices is being addressed,

said integrated circuit operating during a memory-to-I/O transfer or I/O-to-memory transfer to directly transfer data between a memory and an I/O device and to control said external register to not store data for use during said memory-to-I/O transfer or I/O-to-memory transfer,

wherein one or more of said memory devices are connected so as to be controlled by said integrated circuit to read or write data on said bus, wherein said direct memory access unit is contained within a microprocessor chip, and

wherein said microprocessor is a Model V50 microprocessor.

2. The circuit of claim **1** wherein said direct memory access unit, first memory device, and second memory device are connected to a common address bus.

3. The circuit of claim **1** wherein said one or more control terminals of said external register comprise a data input enable terminal and an output enable terminal.

4. The circuit of claim **1** wherein said external register is solely dedicated to said direct memory-to-memory transfer operation.

5. A circuit connected to a direct memory access unit formed as an integrated circuit, wherein said direct memory access unit does not have, internal to said integrated circuit, a temporary storage register for temporarily storing data during a memory-to-memory transfer, said circuit comprising:

a bus directly connected to said integrated circuit;

two or more memory devices connected to said bus in parallel with said integrated circuit; and

an external register, external to said integrated circuit and connected to said bus in parallel with said memory devices, said external register having one or more control terminals coupled to receive control signals generated by said integrated circuit, said external register being controlled by said integrated circuit to temporarily store data only during a direct memory-to-memory transfer operation while one of said memory devices is being addressed,

said integrated circuit operating during a memory-to-I/O transfer or I/O-to-memory transfer to directly transfer data between a memory and an I/O device and to control said external register to not store data for use during said memory-to-I/O transfer or I/O-to-memory transfer,

wherein one or more of said memory devices are connected so as to be controlled by said integrated circuit to read or write data on said bus,

wherein said direct memory access unit is contained within a microprocessor chip, and

wherein said microprocessor is a Model V40 microprocessor.

6. The circuit of claim **5** wherein said direct memory access unit, first memory device, and second memory device are connected to a common address bus.

7. The circuit of claim **5** wherein said one or more control terminals of said external register comprise a data input enable terminal and an output enable terminal.

8. The circuit of claim **5** wherein said external register is solely dedicated to said direct memory-to-memory transfer operation.

* * * * *

US006112274A

United States Patent [19]

Goe et al.

[11] Patent Number: 6,112,274

[45] Date of Patent: Aug. 29, 2000

[54] **METHOD AND APPARATUS FOR PROCESSING MORE THAN ONE INTERRUPTS WITHOUT REINITIALIZING THE INTERRUPT HANDLER PROGRAM**

[75] Inventors: **Richard Goe**, Redwood City; **Vijay Goru**, San Jose; **George Thangadurai**, Santa Clara, all of Calif.

[73] Assignee: **Intel Corporation**, Santa Clara, Calif.

[21] Appl. No.: **09/098,993**

[22] Filed: **Jun. 17, 1998**

[51] **Int. Cl.**[7] **G06F 13/24**; G06F 13/26

[52] **U.S. Cl.** **710/260**; 710/263; 710/264; 710/266; 710/269

[58] **Field of Search** 710/260, 263, 710/264, 269, 47, 48, 261, 265, 268, 266, 267

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,261,107	11/1993	Klim et al.	710/266
5,446,910	8/1995	Kennedy et al.	353/20
5,511,200	4/1996	Jayakumar	710/266
5,675,807	10/1997	Iswandhi et al.	710/260
5,867,687	2/1999	Simpson	710/264
5,896,414	4/1999	Meyer et al.	375/222
5,944,840	8/1999	Lever	714/34

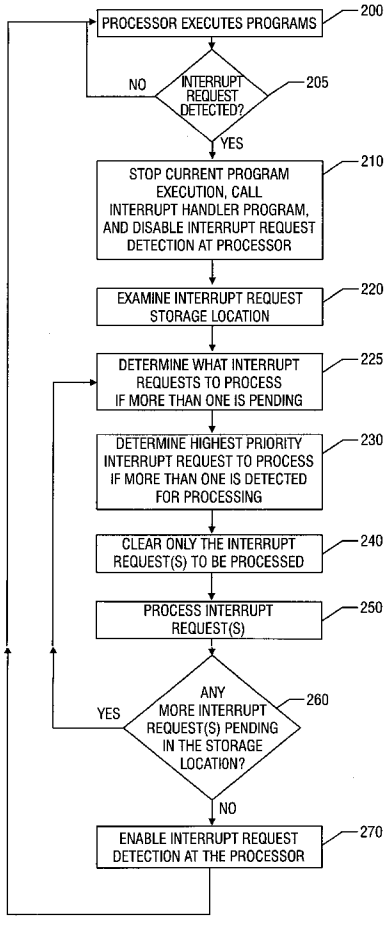
Primary Examiner—Ario Etienne

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman LLP

[57] **ABSTRACT**

A system and method is provided for processing interrupt requests. The method is accomplished by detecting when an interrupt request is being stored in a storage location, examining the storage location storing the interrupt request, prioritizing the interrupt request when more than one interrupt request is stored in the storage location to determine an interrupt request processing order, clearing the interrupt request in the storage location that is to be processed, and processing the interrupt request. The system comprises a first storage location for storing an interrupt request, a second storage location for storing an interrupt handler program with encoded statements for examining the first storage location, prioritizing the interrupt request that is to be processed if more than one interrupt request is stored in said first storage location, clearing the interrupt request that is to be processed, and processing the interrupt request. The system further comprises a processor for executing the interrupt handler program.

19 Claims, 5 Drawing Sheets



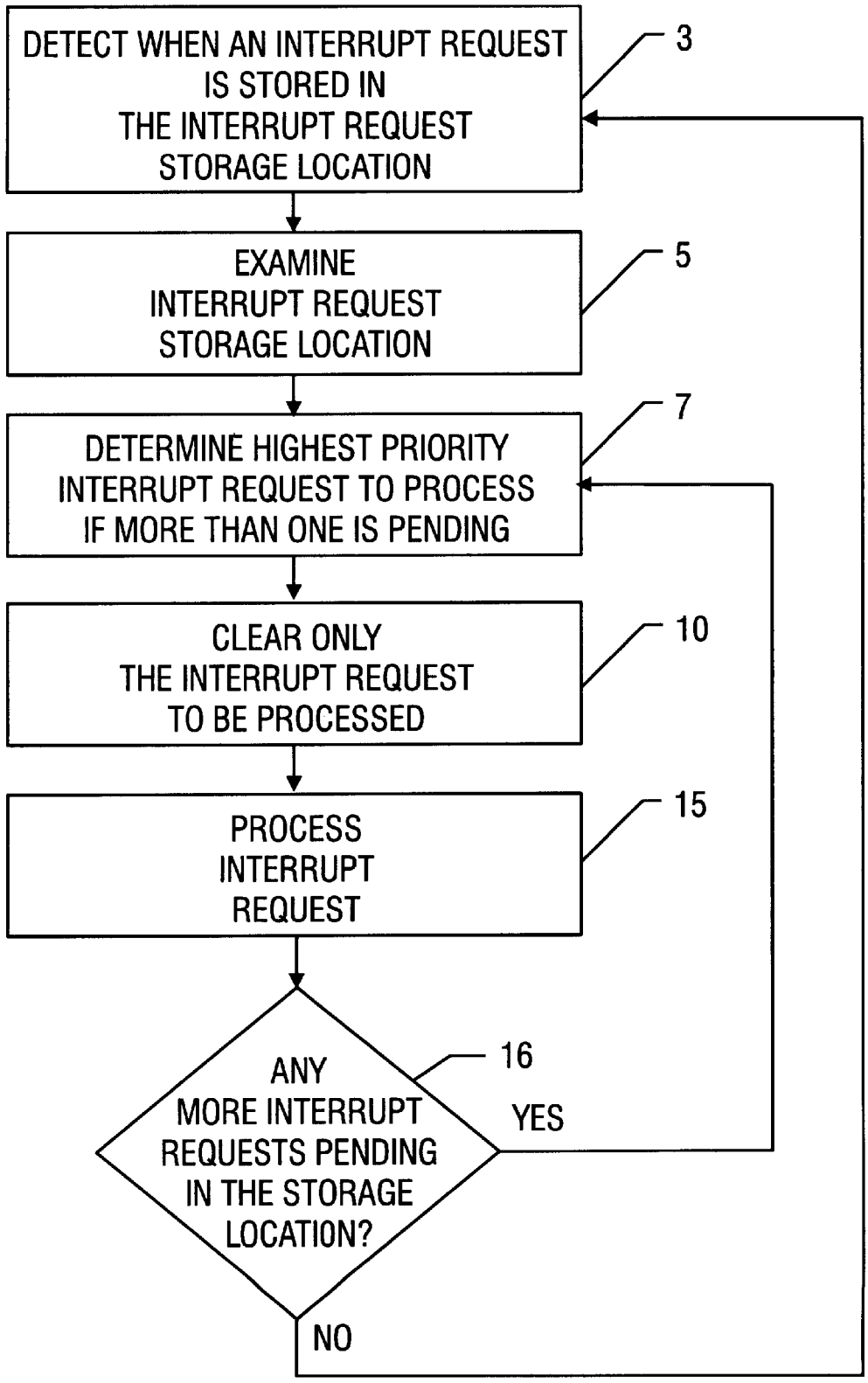

DETECT WHEN AN INTERRUPT REQUEST IS STORED IN THE INTERRUPT REQUEST STORAGE LOCATION
3
EXAMINE INTERRUPT REQUEST STORAGE LOCATION
5
DETERMINE HIGHEST PRIORITY INTERRUPT REQUEST TO PROCESS IF MORE THAN ONE IS PENDING
7
CLEAR ONLY THE INTERRUPT REQUEST TO BE PROCESSED
10
PROCESS INTERRUPT REQUEST
15
ANY MORE INTERRUPT REQUESTS PENDING IN THE STORAGE LOCATION?
16
YES
NO

FIG. 1

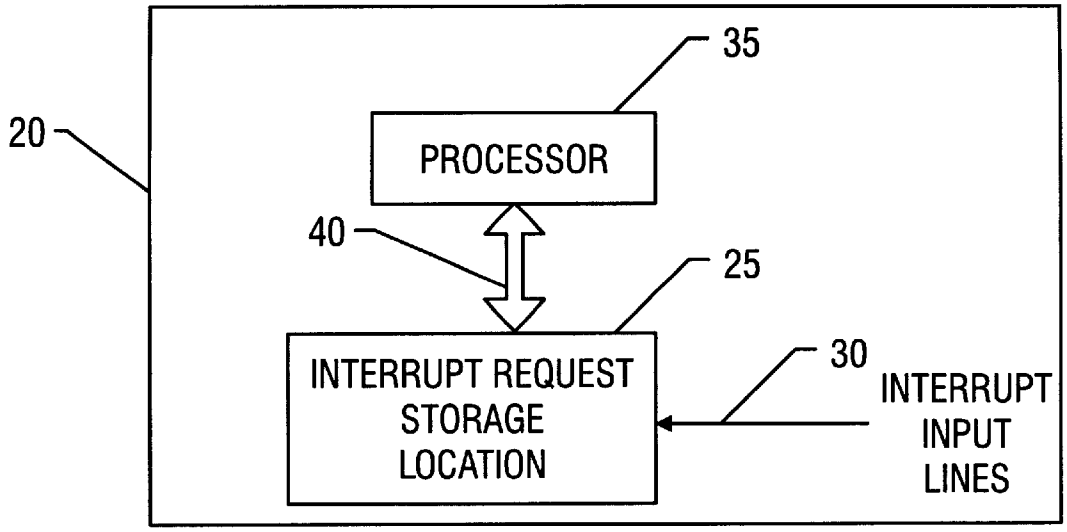


FIG. 2

FIG. 3A

FIG. 3B

FIG. 3

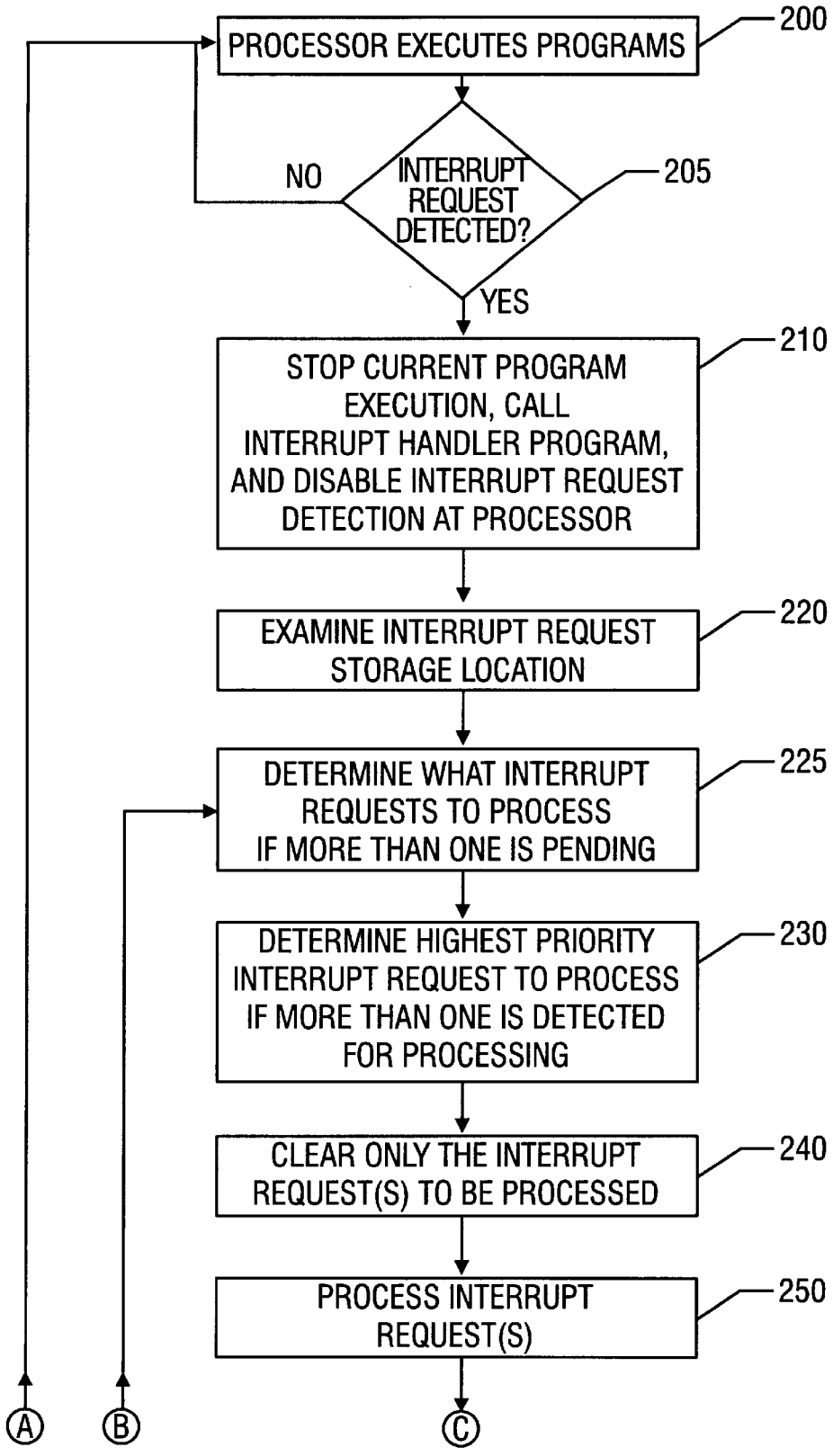

PROCESSOR EXECUTES PROGRAMS
200
INTERRUPT REQUEST DETECTED?
205
NO
YES
STOP CURRENT PROGRAM EXECUTION, CALL INTERRUPT HANDLER PROGRAM, AND DISABLE INTERRUPT REQUEST DETECTION AT PROCESSOR
210
EXAMINE INTERRUPT REQUEST STORAGE LOCATION
220
DETERMINE WHAT INTERRUPT REQUESTS TO PROCESS IF MORE THAN ONE IS PENDING
225
DETERMINE HIGHEST PRIORITY INTERRUPT REQUEST TO PROCESS IF MORE THAN ONE IS DETECTED FOR PROCESSING
230
CLEAR ONLY THE INTERRUPT REQUEST(S) TO BE PROCESSED
240
PROCESS INTERRUPT REQUEST(S)
250
A
B
C

FIG. 3A

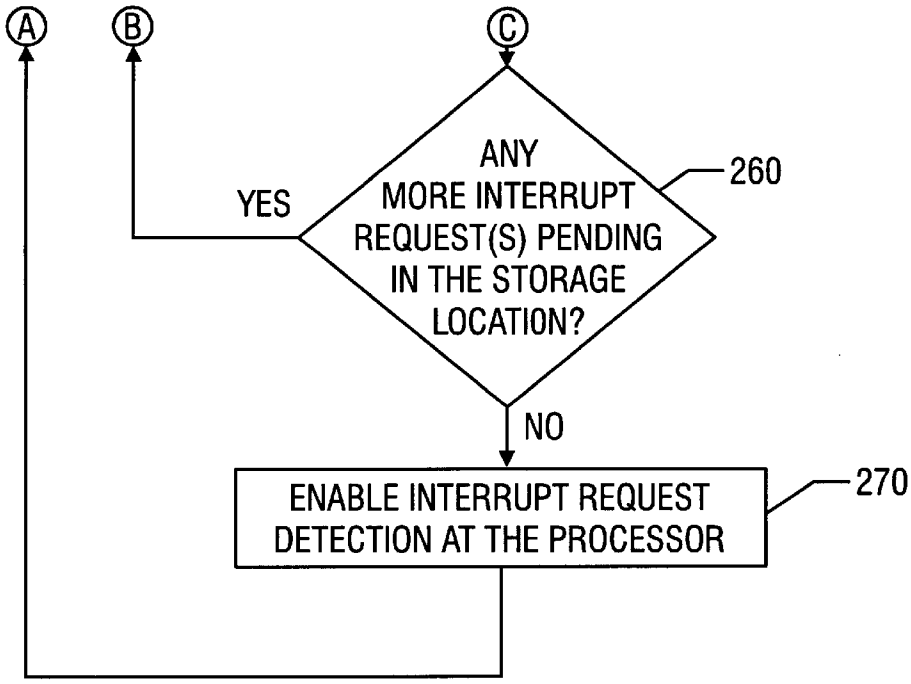

A
B
C
ANY MORE INTERRUPT REQUEST(S) PENDING IN THE STORAGE LOCATION?
260
YES
NO
ENABLE INTERRUPT REQUEST DETECTION AT THE PROCESSOR
270

FIG. 3B

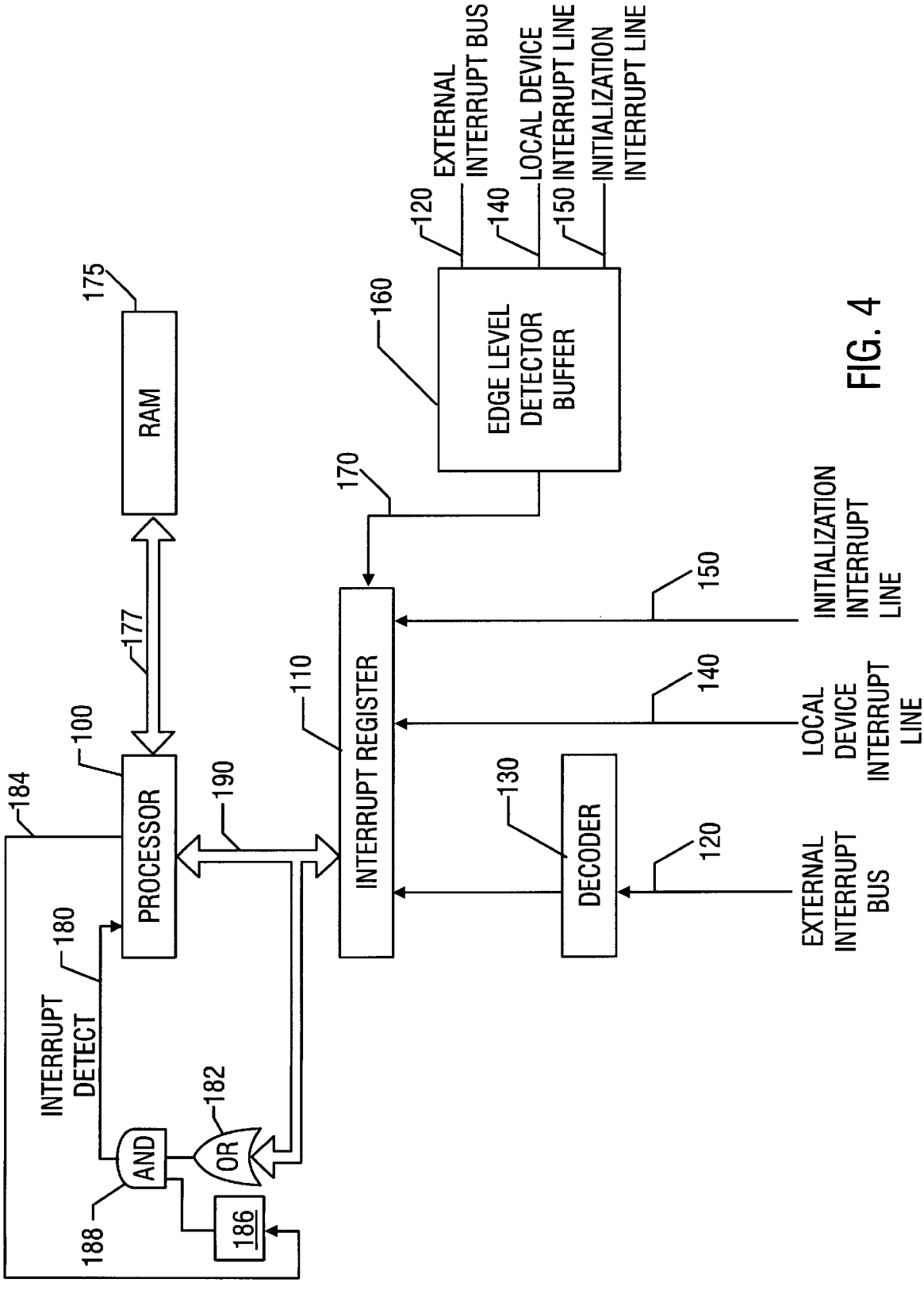

EXTERNAL INTERRUPT BUS
LOCAL DEVICE INTERRUPT LINE
INITIALIZATION INTERRUPT LINE
120
140
150
160
EDGE LEVEL DETECTOR BUFFER
175
RAM
170
177
100
110
INTERRUPT REGISTER
150
INITIALIZATION INTERRUPT LINE
140
LOCAL DEVICE INTERRUPT LINE
130
DECODER
120
EXTERNAL INTERRUPT BUS
184
PROCESSOR
190
180
INTERRUPT DETECT
182
AND
OR
188
186

FIG. 4

METHOD AND APPARATUS FOR PROCESSING MORE THAN ONE INTERRUPTS WITHOUT REINITIALIZING THE INTERRUPT HANDLER PROGRAM

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates generally to the field of data processing systems, and more particularly, to interrupts in data processing systems. Specifically, the present invention relates to improving the processing of interrupts.

2. Description of the Related Art

The demand for quicker and more powerful personal computers has led to many technological advances in the computer industry, including the development of faster processors to run software programs. However, at times it is necessary to stop a processor from running a software program to take care of an exceptional condition presented to the processor in the form of an interrupt request. The condition may be from an external hardware device that requires the use of the processor. There may be several types of interrupts and, to handle an interrupt request, the processor invokes an appropriate program depending on the type of interrupt request received.

Interrupt requests are stored in a status register while the processor determines what types of interrupt requests are pending. The status register has generally been designed to either store a single coded interrupt request or to hold many bits each indicating a type of interrupt request. In either case, when multiple interrupt requests are pending hardware logic must be designed to institute a priority mechanism. To institute a more complicated priority mechanism based on all the types of interrupt requests pending and the type of program that was being run in the processor can lead to substantial hardware. This hardware will occupy more area on the integrated circuit chip or require additional space in an implementation with discrete hardware components.

While an interrupt request is processed, the status register is not modified by the hardware preventing any additional interrupt requests from being taken. This leads to interrupt requests being lost. To prevent interrupt requests from being lost, current designs utilize other secondary status registers to hold these interrupt requests. This solution again requires additional hardware. Furthermore, the procedure is time consuming because interrupt handler software has to read the interrupt requests from multiple hardware elements. Thus, this solution exacerbates the problems associated with a single status register implementation.

SUMMARY OF THE INVENTION

In one aspect of the present invention, a method is provided for handling interrupt requests. The method comprises the steps of detecting when an interrupt request is being stored in a storage location, examining the storage location holding the interrupt request, prioritizing a processing order for the interrupt requests when more than one interrupt request is stored in the storage location, clearing the interrupt request that is to be processed, and processing the interrupt request.

In another aspect of the present invention, a program storage device readable by a general purpose computer and tangibly embodying a program of instructions executable by the computer is provided. The program of instructions detects when an interrupt request is being stored in a storage location, examines the storage location storing an interrupt request, prioritizes the processing order of the interrupt requests when more than one interrupt request is stored in the storage location, clears the interrupt request in the storage location that is to be processed, and processes the interrupt request.

In yet another aspect of the present invention a system for handling interrupt requests is provided. The system for handling interrupt requests comprises a storage location capable of holding an interrupt request, a second storage location storing an interrupt handler program with instructions for examining a storage location holding an interrupt request, prioritizing the processing order of the interrupt requests, clearing the interrupt request in the storage location that is to be processed, and processing the interrupt request. The system further comprises a processor for executing the interrupt handler program.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a flowchart illustrating a method for handling interrupt requests.

FIG. **2** is a high-level diagram illustrating a system for handling interrupt requests.

FIG. **3** interrelates FIG. **3**A and FIG. **3**B.

FIG. **3**A is partial view of a flowchart illustrating a method for handling interrupt requests.

FIG. **3**B is partial view of a flowchart illustrating a method for handling interrupt requests.

FIG. **4** is a block diagram of an embodiment of the system of the present invention.

While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof have been shown by way of example in the drawings and are herein described in detail. It should be understood, however, that the description herein of specific embodiments is not intended to limit the invention to the particular forms disclosed, but on the contrary, the intention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the invention as defined by the appended claims.

DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS

Illustrative embodiments of the invention are described below. In the interest of clarity, not all features of an actual implementation are described in this specification. It will of course be appreciated that in the development of any such actual embodiment, numerous implementation-specific decisions must be made to achieve the developers' specific goals, such as compliance with system-related and business-related constraints, which will vary from one implementation to another. Moreover, it will be appreciated that even if such a development effort might be complex and time-consuming, it would nevertheless be a routine undertaking for those of ordinary skill in the art having the benefit of this disclosure.

Turning to the diagrams, FIG. **1** is a flowchart illustrating a method of handling one or more interrupt requests held in a storage location in a data processing system. As shown in block **3** of FIG. **1**, the method first detects when an interrupt request is stored in an interrupt request storage location. Next, the storage location holding one or more interrupt requests is examined to determine what interrupt requests are pending, as represented in block **5**. If more than one interrupt request is pending, the highest priority interrupt request to process is determined, as exemplified in block **7**.

The prioritization scheme used in block **7** is implementation specific, for example based on the application running in the processor. The interrupt request that is to be processed is cleared from the storage location, as represented in block **10**. Next, as illustrated by block **15**, the interrupt request is processed. Finally, a decision block **16** is reached to determine if any more interrupt requests are pending. If no additional interrupt requests are pending, the method illustrated in FIG. **1**, returns to block **3**, and awaits further detection of interrupt requests. If one or more interrupt requests are still pending then the method returns to block **7**, to determine the highest priority interrupt pending that will be processed and continues as previously described.

As will be appreciated by one of skill in the art having the benefit of this disclosure, decision block **16** permits the processing of recently stored interrupt requests while the method was handling another interrupt request in blocks, **7**, **10** and **15**. Furthermore, the embodiment described in FIG. **1** can be modified to prioritize and process more than a single interrupt request at a time. In this modified embodiment at block **7** multiple high priority interrupts can be selected for processing. At block **10**, only the multiple interrupt requests to be processed are cleared, and at block **15** the multiple interrupt requests are processed.

FIG. **2** is a high-level diagram illustrating an embodiment in accordance with the present invention. FIG. **2** shows a data processing system **20** that contains an interrupt request storage location **25** for holding one or more interrupt requests. The interrupt request storage location **25** holds the interrupt requests as set by external or internal interrupt input lines **30**. The interrupt request storage location **25** can be any type of temporary read/write memory. An interrupt handler program running in the processor **35** examines the interrupt storage location **25**, via a read/write bus **40**, to determine the specific interrupt requests that are pending. The interrupt handler program is a coded embodiment of the interrupt request processing method of the present invention. The interrupt handler program running in the processor **35** prioritizes the order in which the interrupt requests will be processed if more than one interrupt request is pending. The interrupt handler program may also decide not process one or more interrupt requests. The reasons to not process an interrupt request are implementation specific, but can depend for example on the number and type of interrupt requests pending. After deciding what interrupt requests to process the interrupt handler program clears the pending interrupt request(s) in the interrupt request storage location **25** by writing to those locations via the read/write bus **40**. Next, the processor **35** will process the interrupt requests.

FIG. **3**A and FIG. **3**B together show a detailed flow chart of one embodiment of a method in accordance with the present invention. FIG. **3** shows how to interrelate FIG. **3**A and FIG. **3**B. Generally a processor is executing a program as represented in block **200**. The processor will periodically monitor for an interrupt request as represented by decision block **205**. After the processor detects the interrupt request, the processor stops execution of any running program and calls the interrupt handler program as represented in block **210**. During this step the processor will also disable interrupt detection. Detecting interrupt requests and calling an interrupt handler program upon such detection are well known in the art. Calling the interrupt handler program may involve reading the program instructions from RAM or ROM (read only memory) memory. Any suitable technique known to the art may be employed and the technique employed by a particular embodiment will be implementation specific. Next, the interrupt handler program examines an interrupt request storage location as represented in block **220**. In one particular embodiment, the interrupt request storage location is a storage unit, known as a register, having several bits indicating whether any interrupt requests are pending, and the types of interrupt requests that are pending. However, the invention is not so limited, as other types of storage mechanisms may be used. Other aspects, including the number and types of interrupt requests that can be held in the interrupt storage location are implementation dependent. According to the present invention, more than one interrupt request may be stored in the interrupt storage location.

Next, as illustrated in block **225**, the interrupt handler program determines which of the pending interrupt requests it will handle if more than one interrupt request is stored in the interrupt storage location. However, the method of the present invention is flexible, and thus this step is optional based on the specific implementation.

Next, as represented in block **230**, if the interrupt handler program determines to process more than a one interrupt request, the interrupt handler program prioritizes the processing order for the interrupt requests selected for processing in block **225**. When the step represented by block **225** is not part of the embodiment, the interrupt handler program prioritizes and selects one interrupt request to process at block **230**. The prioritization rules employed by a particular embodiment may be implementation specific. Typically, certain types of interrupt requests, such as those intended to initialize a processor, will be assigned higher priority than others and the interrupts are processed according to these assigned priorities.

The next step, as represented in block **240**, is to clear within the storage location only the interrupt requests that the interrupt handler program will process. Other pending interrupt requests are not cleared. As represented in block **250**, the interrupt handler program will next process the interrupt request(s) as determined in block **230**. This embodiment allows for processing more than one interrupt request if the optional step in block **225** is supported.

Next, as represented by decision block **260**, after processing the interrupt requests the interrupt handler program will re-examine the interrupt request storage location to determine if any more interrupt requests are pending. If one or more interrupt requests are still pending then the method loops back to block **225** to determine what interrupt requests to process if more than one interrupt request is pending. If the embodiment does not have the optional step represented by block **225**, the method will loop back to block **230** to prioritize and determine what interrupt request to process if more than one interrupt request is pending. One skilled in the art with the benefit of this disclosure would appreciate that more interrupt requests can be stored in the interrupt storage location while the method of the present embodiment goes through blocks **225**, **230**, **240**, and **250**. In this case, these additional interrupt requests are detected at decision block **260**. If no pending interrupt requests are found at decision block **260**, then the interrupt request detection is enabled again, as represented by block **270**.

Finally, the method returns to block **200** to resume execution of the program being processed prior to the initial interrupt request being detected. As will be appreciated by those skilled in the art having the benefit of this disclosure, once the processor resumes execution it is possible to again receive one or more interrupt requests.

FIG. **4** is a block diagram of one embodiment of an apparatus that can be used to implement the method of FIG. **3**A and FIG. **3**B. This particular embodiment comprises a

processor **100** that may execute programs, including the interrupt handler program, and detect interrupt requests. The embodiment also comprises an interrupt storage location, that in this particular embodiment is an external register, the interrupt register **110**. However, the interrupt storage location may be external RAM or internal memory in the processor **100** based on various alternative embodiments.

As will be appreciated by those skilled in the art, a processor can receive many types of interrupt requests. Some of those interrupt requests are those received from external hardware devices. A processor may also generally receive local device interrupts and an initialization interrupt. In the particular embodiment illustrated in FIG. **4**, external interrupts are vectored and coded into 4 bits and must be decoded prior to storage in the interrupt register **110**. The interrupt register **110** is a 64 bit register that may be set through individual pins and is read/write accessible through a control register access bus **190**. The size of the interrupt register **110** may vary according to the data processing system that is implemented.

FIG. **4** shows the external interrupt bus **120** connected to a 4-to-16 bit decoder **130**. The decoded vector of 16 bits is input to the interrupt register **110**. A local device interrupt line **140** is directly connected to the interrupt register **110**. An initialization interrupt line **150** is also directly connected to the interrupt register **110**. As will be appreciated by those skilled in the art the number of external interrupts or local device interrupts may vary according to the application. Likewise, the external interrupts or local device interrupts may or may not be encoded in various alternative embodiments.

According to the present embodiment, the interrupt register **110** will store the output bits from the decoder **130**, from the local device interrupt line **140**, or from the initialization interrupt line **150**, as determined by the output of an edge level detector buffer **160**. The edge level detector buffer **160** detects whether an interrupt has been received by detecting either a rise or fall of an edge, or both the rise and fall of an edge depending on the settings of the edge level detector buffer **160**. The edge level detector buffer will output a register control bit **170**, used to register the bits of the interrupt inputs into the interrupt register **110**.

The processor **100** reads and executes the interrupt handler program whenever the interrupt detect signal **180** is set. The interrupt detect signal **180** is partially controlled by the processor **100** with an interrupt enable signal **184**. The interrupt enable signal **184** output by the processor **100** is stored in a single bit register **186**. The interrupt detect signal **180** is determined by performing an "AND" function **188** of the interrupt enable signal **184** stored in the single bit register **186** with the output of the "OR" function **182** performed on the access bus bits **190**. The access bus bits contain any set bits indicating that an interrupt request has been stored in the interrupt register **110**. Therefore, if at least one interrupt request has been stored in the interrupt register **110** and the processor has enabled the interrupt enable signal **184**, the interrupt detect signal **180** is set.

The interrupt handler program that follows the method described in FIG. **3**A and FIG. **3**B will be read from RAM memory **175** when the interrupt detect signal **180** is set and detected by processor **100**. As will be appreciated by those skilled in the art the interrupt handler program may be encoded on different storage program medium types, such as ROM memory based on the specific embodiment. The processor **100** will read the interrupt handler program over RAM bus **177**.

Once executing, the interrupt handler program uses the control register access bus **190** to read into the interrupt register **110** and determine what interrupt requests have been stored and are pending. The interrupt handler program according to the method shown in FIG. **3**A and FIG. **3**B, can determine what interrupt requests to process and prioritize the interrupt processing schedule. The interrupt handler program will also write to the interrupt register **110** over the control register access bus **190** to clear the bits of the interrupt requests that are to be processed. As one skilled in the art can appreciate, the interrupt handler program can be programmed to flexibly handle the interrupts based on the particular implementation.

Thus, the embodiments of the methods and systems described flexibly handle one or more interrupt requests according to the specific implementation, while increasing the speed for handling the interrupt requests and minimizing the need for additional hardware.

The particular embodiments disclosed above are illustrative only, as the invention may be modified and practiced in different but equivalent manners apparent to those skilled in the art having the benefit of the teachings herein. Furthermore, no limitations are intended to the details of construction or design herein shown, other than as described in the claims below. It is therefore evident that the particular embodiments disclosed above may be altered or modified and all such variations are considered within the scope and spirit of the invention. Accordingly, the protection sought herein is as set forth in the claims below.

What is claimed:

1. A method for handling interrupts comprising:

detecting when interrupts occur;

storing into a storage location a vector of bits wherein each bit is associated with an interrupt;

calling an interrupt handler program to process the interrupts;

examining the storage location to identify presence of interrupts which are to be processed by the interrupt handler program;

prioritizing those interrupts which are to be processed by the interrupt handler program;

clearing a bit in the storage location, associated with an interrupt request being processed;

processing the interrupt request; and

processing subsequent interrupt requests which the interrupt handler program can process by clearing an appropriate bit in the storage location for that interrupt and processing a next prioritized interrupt request without reinitializing the interrupt handler program.

2. The method of claim **1**, further comprising enabling an interrupt enable bit by a processor to initiate a reading of the storage location to identify if any interrupt is present.

3. The method of claim **1**, further comprising updating said storage location with new interrupt requests to be processed by the interrupt handler program, while the interrupt handler program is processing a previous interrupt request, the interrupt handler program then processing the new interrupt without reinitializing the interrupt handler program.

4. A method for handling interrupts comprising:

detecting when interrupts occur;

storing into a storage location a vector of bits wherein each bit is associated with an interrupt:

generating an interrupt enable signal to indicate that interrupt processing is to be performed;

calling an interrupt handler program to process the interrupts;

examining the storage location to identify presence of interrupts which are to be processed by the interrupt handler program, if the interrupt enable signal is set to indicate that interrupt processing is to be performed;

prioritizing those interrupts which are to be processed by the interrupt handler program;

clearing a bit in the storage location, associated with an interrupt request being processed;

processing the interrupt request; and

processing subsequent interrupt requests which the interrupt handler program can process by clearing an appropriate bit in the storage location for that interrupt and processing a next prioritized interrupt request without reinitializing the interrupt handler program, provided the interrupt enable signal is still set.

5. The method of claim **4**, further comprising enabling an interrupt enable bit by a processor to generate the interrupt enable signal.

6. The method of claim **4**, further comprising updating said storage location with new interrupt requests to be processed by the interrupt handler program, while the interrupt handler program is processing a previous interrupt request, the interrupt handler program then processing the new interrupt without reinitializing the interrupt handler program.

7. A program storage medium readable by a general purpose computer and tangibly embodying a program of instructions, which when executed by said computer, performs the following:

detecting when an interrupt occurs;

storing into a storage location a vector of bits wherein each bit is associated with an interrupt;

calling an interrupt handler program to process the interrupts;

reading the storage location to identify presence of interrupts which are to be processed by the interrupt handler program;

prioritizing those interrupts which are to be processed by an interrupt handler program;

clearing a bit in the storage location, associated with the interrupt request being processed;

processing the interrupt request; and

processing subsequent interrupt requests which the interrupt handler program can process by clearing an appropriate bit in the storage location for that interrupt and processing a next prioritized interrupt request without reinitializing the interrupt handler program.

8. The program storage medium of claim **7**, wherein the program of instructions further comprise enabling an interrupt enable bit by the computer to initiate a reading of the storage location to identify if any interrupt is present.

9. The program storage medium of claim **7**, wherein the program of instructions further comprise updating the storage location with new interrupt requests to be processed by the interrupt handler program, while the interrupt handler program is processing a previous interrupt request, the interrupt handler program then processing the new interrupt without reinitializing the interrupt handler program.

10. A program storage medium readable by a general purpose computer and tangibly embodying a program of instructions, which when executed by said computer, performs the following:

detecting when interrupts occur;

storing into a storage location a vector of bits wherein each bit is associated with an interrupt;

generating an interrupt enable signal to indicate that interrupt processing is to be performed;

calling an interrupt handler program to process the interrupts;

examining the storage location to identify presence of interrupts which are to be processed by the interrupt handler program, if the interrupt enable signal is set to indicate that interrupt processing is to be performed;

prioritizing those interrupts which are to be processed by the interrupt handler program;

clearing a bit in the storage location, associated with an interrupt request being processed;

processing the interrupt request; and

processing subsequent interrupt requests which the interrupt handler program can process by clearing an appropriate bit in the storage location for that interrupt and processing a next prioritized interrupt request without reinitializing the interrupt handler program, provided the interrupt enable signal is still set.

11. The program storage medium of claim **10**, wherein the program of instructions further comprise enabling an interrupt enable bit by the computer to generate the interrupt enable signal.

12. The program storage medium of claim **10**, wherein the program of instructions further comprise updating the storage location with new interrupt requests to be processed by the interrupt handler program, while the interrupt handler program is processing a previous interrupt request, the interrupt handler program then processing the new interrupt without reinitializing the interrupt handler program.

13. An apparatus for handling interrupt requests comprising,

a first storage device for storing bits corresponding to interrupt requests;

a second storage device for storing an interrupt handler program; and

a processor coupled to said second storage device for executing the interrupt handler program, said processor also coupled to said first storage device to read bits stored in said first storage device and prioritizing those interrupts that are to be processed by the interrupt handler program, said processor processing an interrupt after clearing its bit in the first storage device and processing subsequent interrupt requests which the interrupt handler program can process by clearing an appropriate bit in the first storage device for that interrupt to process a next prioritized interrupt request without reinitializing the interrupt handler program.

14. The apparatus of claim **13**, further comprising a third storage location coupled to said processor for storing an interrupt enable bit, which is used by said processor to enable the interrupt handler program.

15. The apparatus of claim **13**, wherein said first storage location is a register.

16. The apparatus of claim **13**, wherein said first storage location is updated while said processor executes the interrupt handler program.

17. A computer system for handling interrupt requests comprising,

first means for storing bits corresponding to interrupt requests;

second means for storing an interrupt handler program; and

means for executing the interrupt handler program, wherein which the bits in the first means for storing are read to identify which interrupts are present and prioritized by the interrupt handler program for those interrupts that are to be processed by the interrupt handler program, said means for executing clearing a corresponding bit associated with an interrupt to be processed by the interrupt handler program processing the interrupt and further processing subsequent interrupts by clearing an appropriate bit as that interrupt is processed by the interrupt handler program, but without reinitializing the interrupt handler program for processing the subsequent interrupts, said means for executing coupled to said first and second means for storing.

18. The computer system of claim **17**, further comprising a third means for storing coupled to said means for executing to store an interrupt enable bit which is used to enable the interrupt handler program.

19. The computer system of claim **17**, wherein said first storing means is updated while the interrupt handler program is processing an interrupt.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,112,274
DATED : August 29, 2000
INVENTOR(S) : Goe et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 7,
Lines 44-45, delete “the interrupt” and insert -- an interrupt --.

Column 9,
Line 2, delete “which”.

Signed and Sealed this

Third Day of September, 2002

Attest:

Attesting Officer

JAMES E. ROGAN
Director of the United States Patent and Trademark Office

US006223196B1

(12) **United States Patent**
Hattori et al.

(10) **Patent No.: US 6,223,196 B1**
(45) **Date of Patent: Apr. 24, 2001**

(54) **SHARED MAC (MULTIPLY ACCUMULATE) SYSTEM AND METHOD**

(75) Inventors: **Tetsuo Hattori**; **Yasuhiro Matsumoto**, both of Shiga-ken (JP)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/074,941**

(22) Filed: **May 8, 1998**

(30) **Foreign Application Priority Data**

Aug. 29, 1997 (JP) 9-233741

(51) **Int. Cl.**[7] **G06F 7/38**
(52) **U.S. Cl.** **708/523**
(58) **Field of Search** 708/523, 501; 712/221

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,791,590	*	12/1988	Ku et al.	708/523
5,204,828	*	4/1993	Kohn	708/501
5,457,804	*	10/1995	Ohtomo	708/523
5,771,185	*	6/1998	Miyake et al.	708/523
6,148,395	*	11/2000	Dao et al.	712/221

* cited by examiner

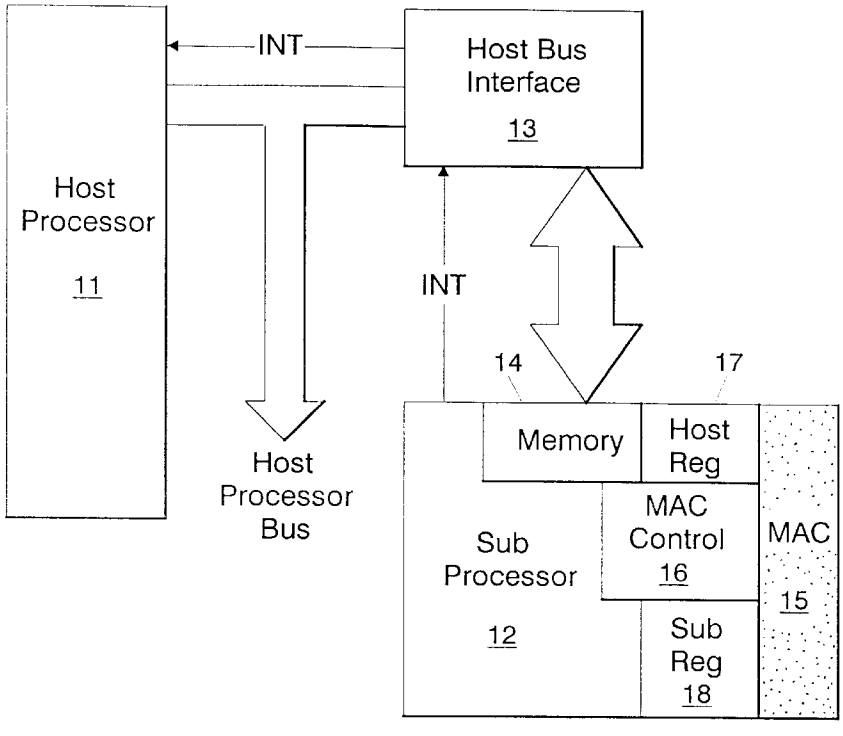
Primary Examiner—David H. Malzahn
(74) *Attorney, Agent, or Firm*—Scully, Scott, Murphy & Presser; Marian Underweiser, Esq.

(57) **ABSTRACT**

The present invention provides for the sharing of a MAC unit included in a sub-processor by the sub-processor and a host processor, so that the period of time for the operation of the host processor is reduced, and the overall performance of the integrated dual processor module is enhanced.

More particularly, the present invention is directed to a shared multiply accumulate (MAC) system comprising: a host processor **11**; a sub-processor **12** including multiply accumulate (MAC) unit **15**; a host bus interface **13** forming a connection between the host processor **11** and the sub-processor **12**; a first register set **17** in which a multiplier, a multiplicand, a product and a status are written in order for the performance of a multiply accumulate (MAC) operation required by the host processor **11**; a second register set **18** in which a multiplier, a multiplicand, a product and a status are written in order for the performance of a multiply accumulate (MAC) operation required by the sub-processor **12**; and selection means for selecting the first register set **17** or the second register set **18** upon receipt of a frequency division signal obtained by the division of an internal clock in the shared MAC system, wherein the MAC unit **15** in the sub-processor **12** performs the MAC operation in accordance with the contents of the first register set **17** or of the second register set **18** that is selected.

7 Claims, 5 Drawing Sheets



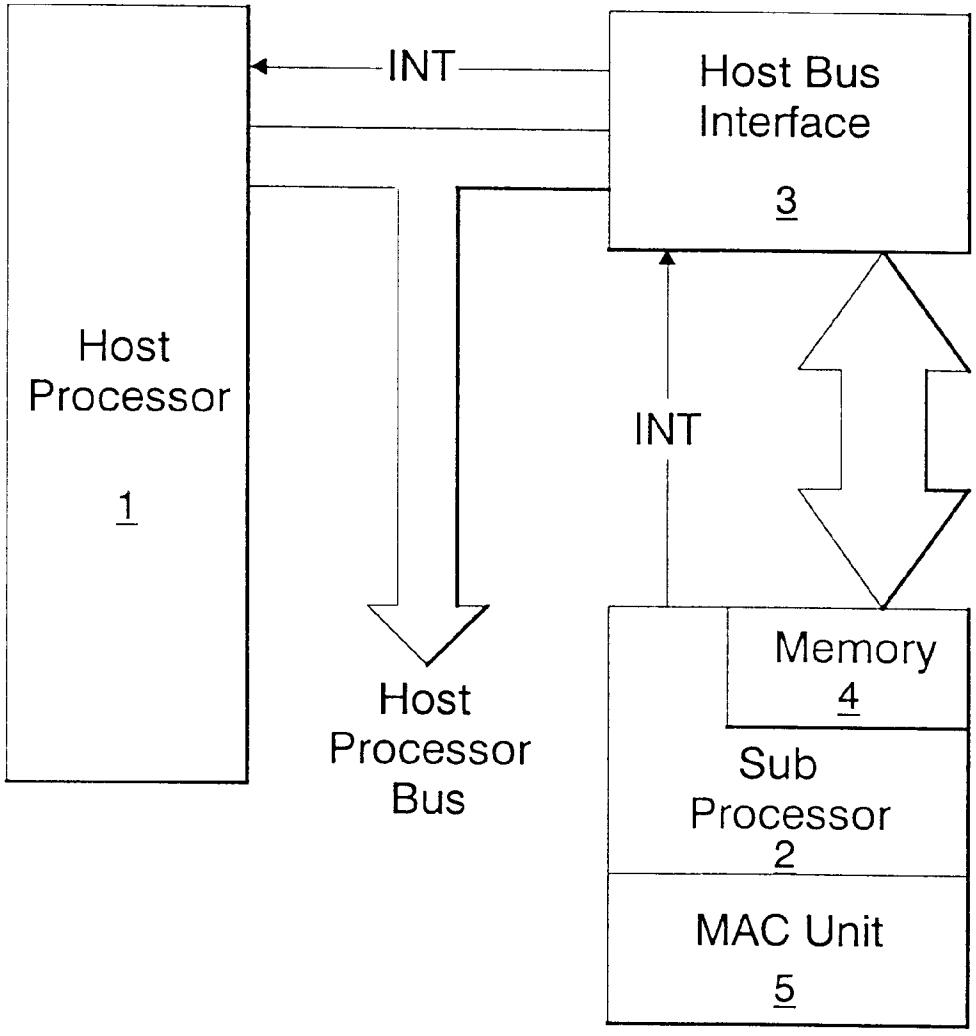

INT
Host Bus Interface
3
Host Processor
1
INT
Host Processor Bus
Memory
4
Sub Processor
2
MAC Unit
5

FIG. 1

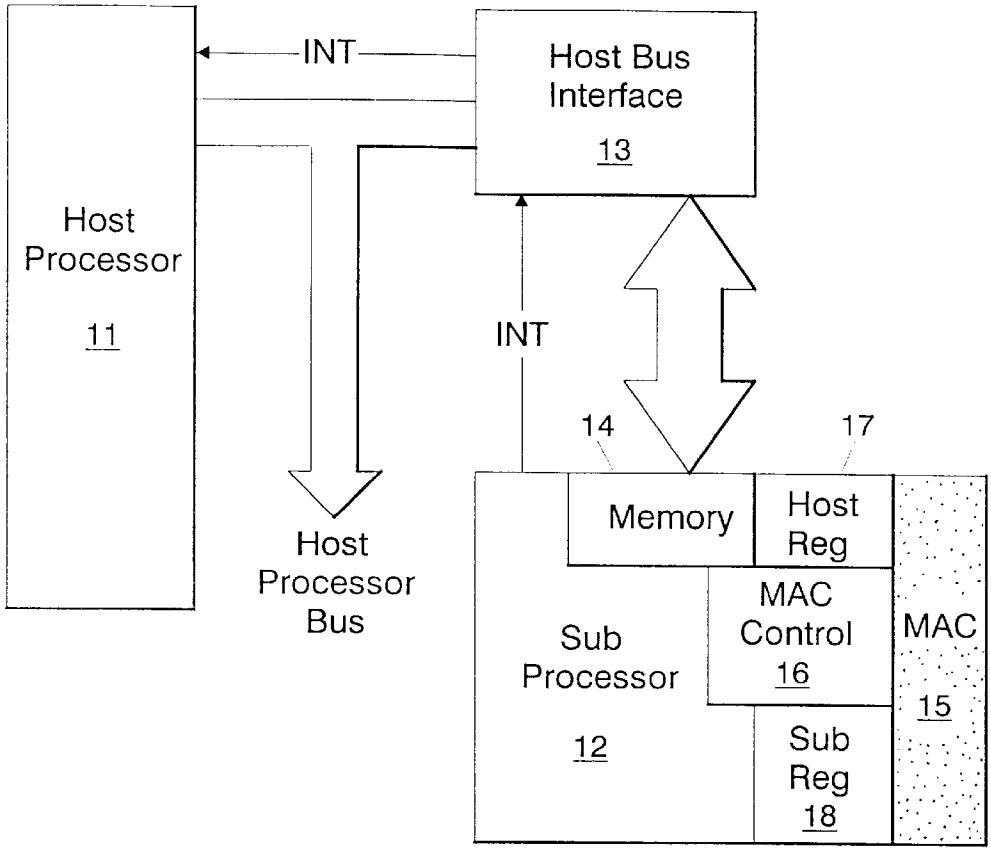

Host Processor
11
INT
Host Bus Interface
13
INT
Host Processor Bus
14
17
Memory
Host Reg
MAC Control
16
Sub Processor
12
Sub Reg
18
MAC
15

FIG. 2

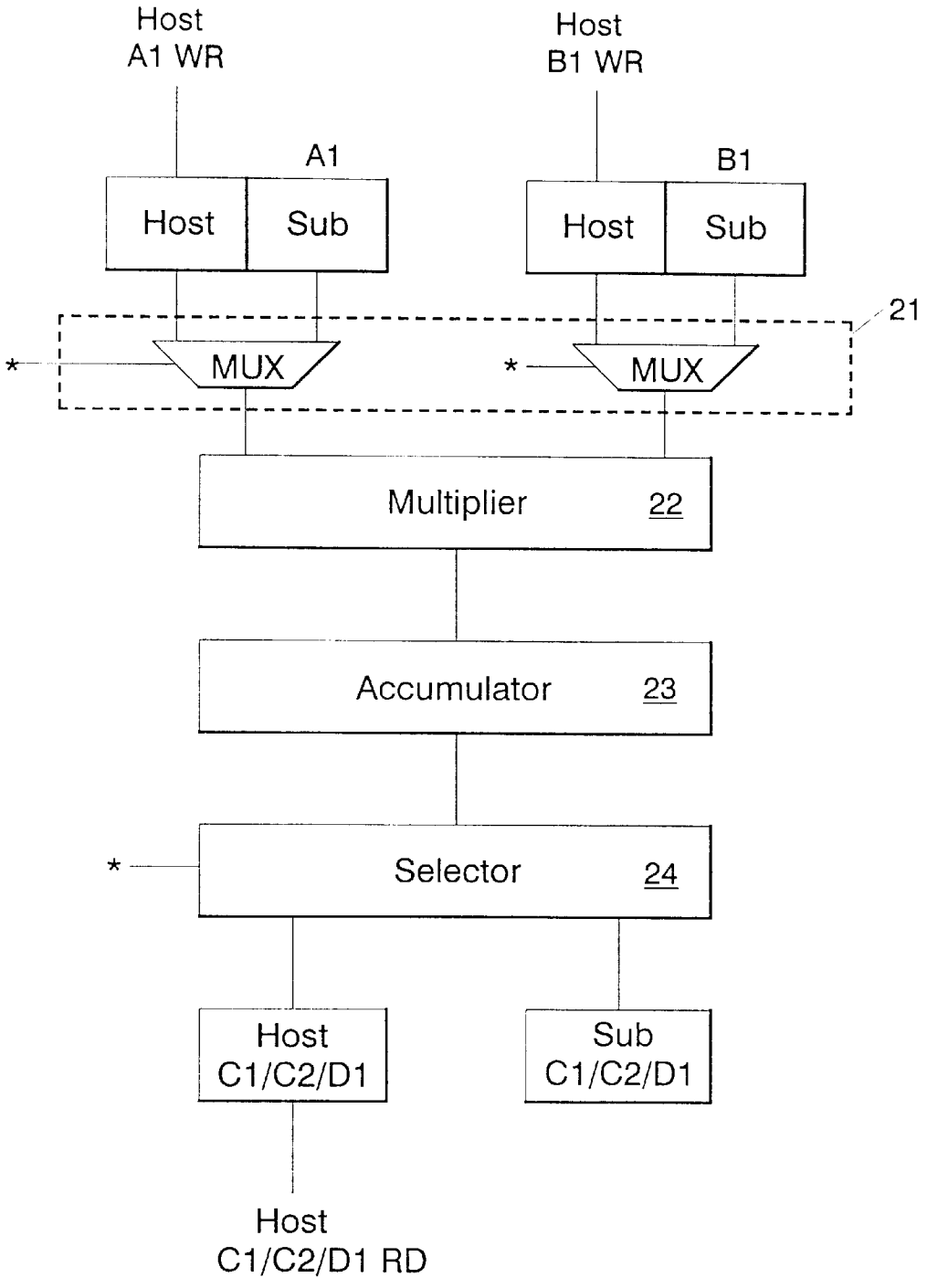

Host
A1 WR
Host
B1 WR
A1
B1
Host
Sub
Host
Sub
21
*
MUX
*
MUX
Multiplier
22
Accumulator
23
*
Selector
24
Host
C1/C2/D1
Sub
C1/C2/D1
Host
C1/C2/D1 RD
*: MAC Sharing

FIG. 3

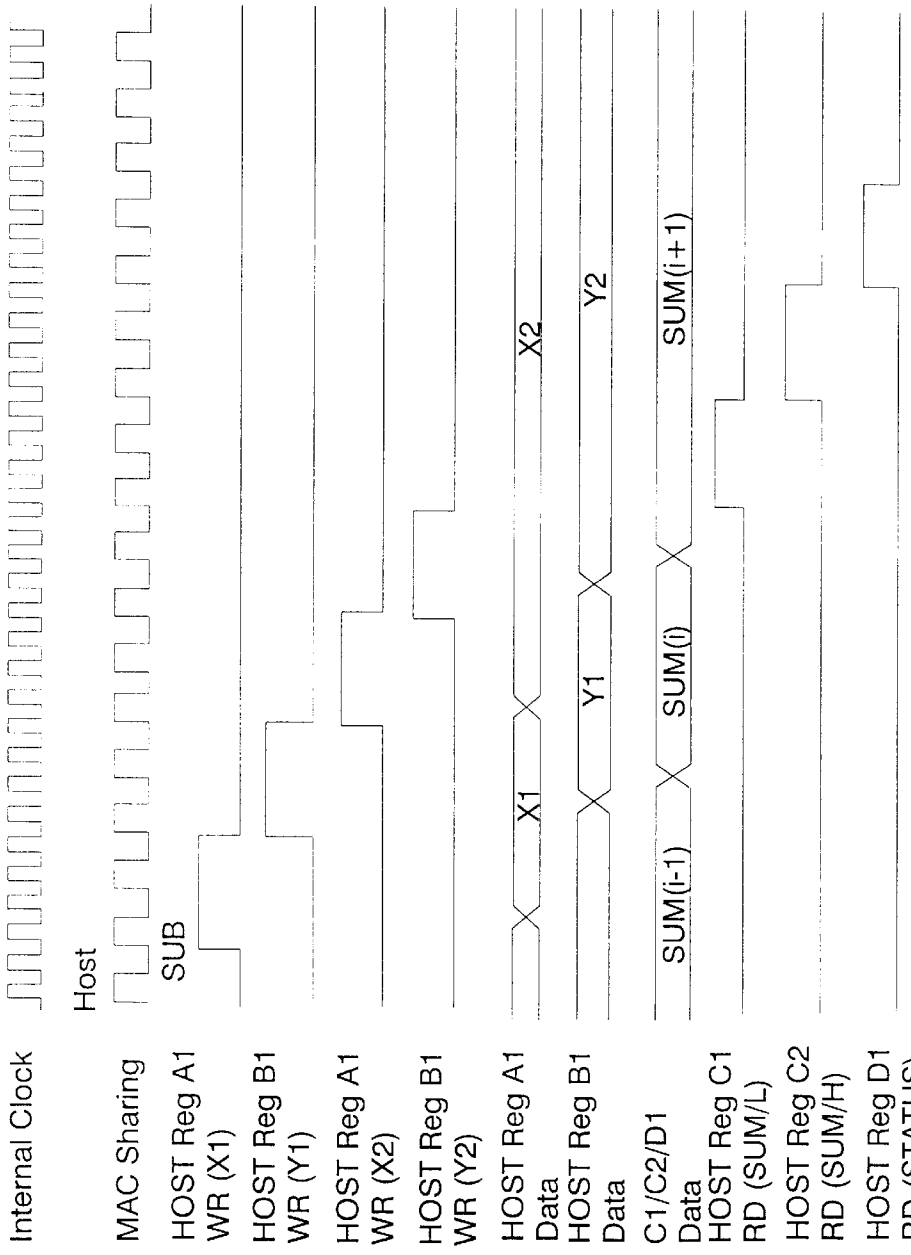

Internal Clock
MAC Sharing
Host
SUB
HOST Reg A1
WR (X1)
HOST Reg B1
WR (Y1)
HOST Reg A1
WR (X2)
HOST Reg B1
WR (Y2)
HOST Reg A1
Data
X1
X2
HOST Reg B1
Data
Y1
Y2
C1/C2/D1
Data
SUM(i-1)
SUM(i)
SUM(i+1)
HOST Reg C1
RD (SUM/L)
HOST Reg C2
RD (SUM/H)
HOST Reg D1
RD (STATUS)

FIG. 4

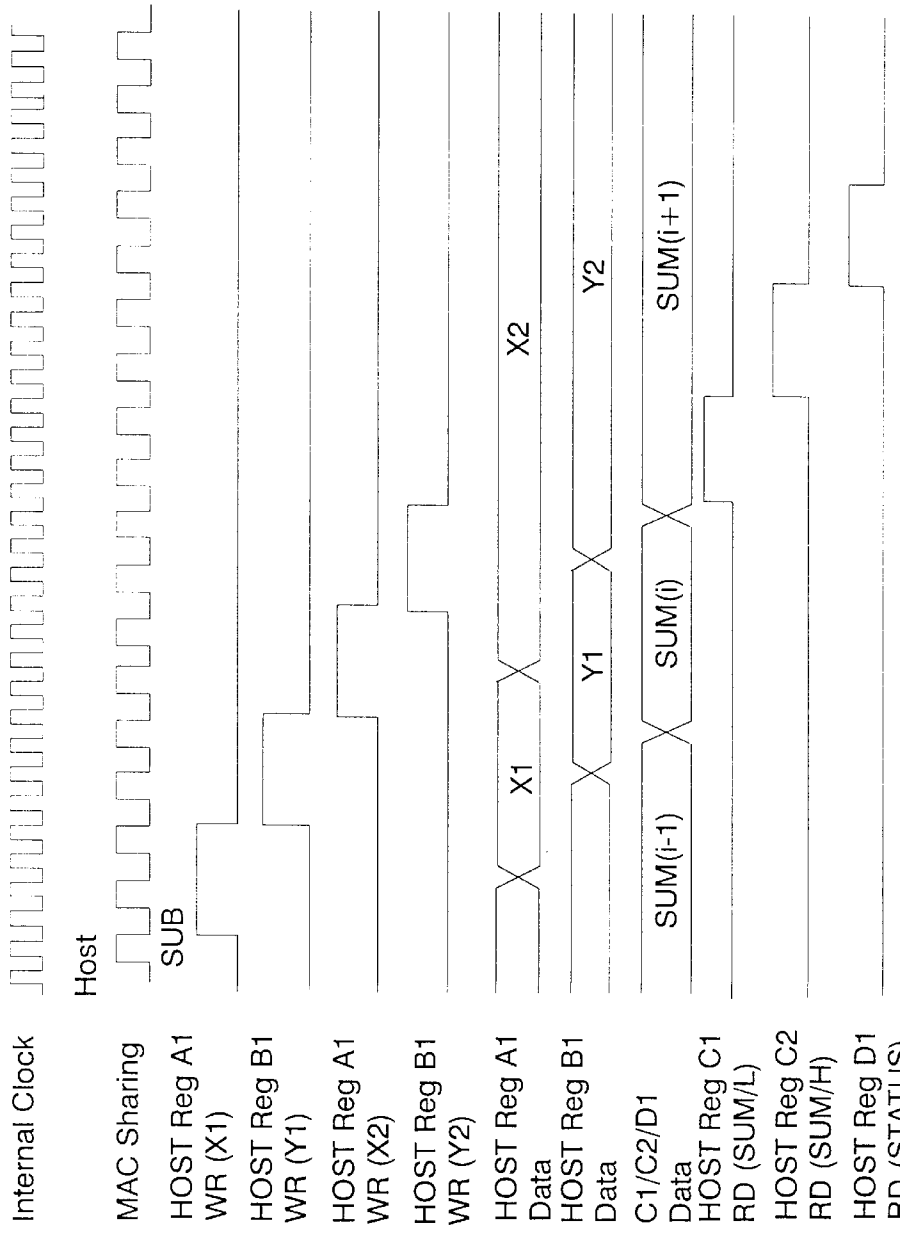

Internal Clock
Host
SUB
MAC Sharing
HOST Reg A1
WR (X1)
HOST Reg B1
WR (Y1)
HOST Reg A1
WR (X2)
HOST Reg B1
WR (Y2)
HOST Reg A1
Data
X1
X2
HOST Reg B1
Data
Y1
Y2
C1/C2/D1
Data
SUM(i-1)
SUM(i)
SUM(i+1)
HOST Reg C1
RD (SUM/L)
HOST Reg C2
RD (SUM/H)
HOST Reg D1
RD (STATUS)

FIG. 5

SHARED MAC (MULTIPLY ACCUMULATE) SYSTEM AND METHOD

BACKGROUND OF THE INVENTION

1. Technical Field

The present invention relates to a shared MAC (Multiply Accumulate) system and a method therefor, and in particular to a method whereby a MAC unit of a sub-processor is employed in common by the sub-processor and a host processor.

2. Prior Art

For a low end application, an integrated dual processor module that provides a high performance at low cost has been proposed for which a host processor and a sub-processor are integrated in a single chip and are independently operated. A host processor can be, for example, a CISC microcontroller for controlling the entire apparatus, and a sub-processor can be, for example, a fuzzy processor for performing a DSP function.

FIG. **1** is a block diagram illustrating the arrangement of a conventional integrated dual processor module. A CISC host processor **1** and a sub-processor **2** are connected by a host bus interface **3**. In the sub-processor **2** are provided a memory **4** and a multiply accumulate (MAC) unit **5**, which performs the important function required of the sub-processor **2**. The MAC unit **5** of the sub-processor **2** normally performs a routine process, such as servo control for which repetitious MAC operations are required, while the microprocessor **1** is primarily responsible for overall system control. The exchange of data by means of interrupt signals transmitted via a memory or a register that is used in common has been proposed as a method by which the two processors can communicate.

In the conventional system, if a single MAC operation must be performed in a processing sequence handled by the host processor **1**, either a multiplication instruction and an addition instruction provided for the host processor **1** must be combined to perform the MAC operation, or interrupt processing must be performed to assign the execution of the MAC operation to the MAC unit **5** in the sub-processor **2**. In either case, however, a relatively long execution time is required for the processing. Particularly in the latter case, where the host processor **1** employs the interrupt process to assign the MAC operation to the sub-processor **2**, the time required for the interrupt processing constitutes a bottleneck for its execution. Therefore, there is a demand for the development of a more efficient system whereby the MAC unit **5** in the sub-processor **2** can be employed in common by the sub-processor **2** and the host processor **1** without any interrupt processing being required.

It is one object of the present invention to improve the operational performance of an integrated dual processor module by reducing the period of time required for the processing performed by a host processor.

It is another object of the present invention to provide an efficient system whereby a Multiply Accumulate (MAC) unit in a sub-processor can be employed in common by the sub-processor and a host processor.

SUMMARY OF THE INVENTION

To achieve the above objects, according to a first aspect of the present invention, a shared multiply accumulate (MAC) system comprises: a host processor; a sub-processor including multiply accumulate (MAC) unit; a host bus interface forming a connection between the host processor and the sub-processor; a first register set in which a multiplier, a multiplicand, a product and a status are written in order for the performance of a multiply accumulate (MAC) operation required by the host processor; a second register set in which a multiplier, a multiplicand, a product and a status are written in order for the performance of a multiply accumulate (MAC) operation required by the sub-processor; and selection means for selecting the first register set or the second register set upon receipt of a frequency division signal obtained by the division of an internal clock in the shared MAC system, wherein the MAC unit in the sub-processor performs the MAC operation in accordance with the contents of the first register set or of the second register set that is selected.

The selection means may include first selection means for, in response to the frequency division signal, supplying to the MAC unit a multiplier and a multiplicand written either in the first register set or in the second register set, and second selection means for, in response to the frequency division signal, supplying either to the first register set or to the second register set a product obtained by the MAC unit and a status.

The MAC unit is constituted by a multiplication unit and an addition unit.

It is preferable that the first and the second register sets be provided in the sub-processor.

According to a second aspect of the present invention, provided is a multiply accumulate (MAC) method for a shared MAC system that includes a host processor, a sub-processor having a multiply accumulate (MAC) unit, a host bus interface connected between the host processor and the sub-processor, and a register set, the MAC method comprising the steps of: writing a multiplier and a multiplicand to a register set via the host bus interface so that a multiply accumulate (MAC) operation required by the host processor can be performed; permitting the MAC unit, in response to a frequency division signal obtained by dividing an internal clock in the shared MAC system, to perform the MAC operation in a time sharing manner based on the multiplier and the multiplicand written in the register set; writing in the register set a product and a status obtained as a result of the MAC operation; and transmitting the product and the status via the host bus interface to the host processor.

The sub-processor, of which a high-speed repetitious MAC operation function is required, does not perform the MAC operation very frequently in general because the frequency at which the repetitious servo control function and so on are employed is relatively low. Thus, the MAC unit provided for the sub-processor is used in common in a time sharing manner as an external operating unit for the host processor. To do this, a host processor register set and a sub-processor register set are selectively connected to the MAC unit in accordance with a frequency division signal obtained by dividing an internal clock in the shared MAC system, and the MAC operation is performed.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the present invention will now be described, by way of example only, with reference to the accompanying drawings in which:

FIG. **1** is a block diagram illustrating the arrangement of a conventional integrated dual processor module.

FIG. **2** is a block diagram illustrating the arrangement of a integrated dual processor module according to one embodiment of the present invention.

FIG. **3** is a block diagram illustrating a multiply accumulate (MAC) unit **15** in the embodiment.

FIG. **4** is a timing chart of the state when the MAC operation is switched by a ½ frequency division signal.

FIG. **5** is a timing chart of the state when the MAC operation is performed with one clock cycle delay.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS OF THE INVENTION

FIG. **2** is a block diagram illustrating the arrangement of an integrated dual processor module in one embodiment of the present invention. A host processor **11** and a sub-processor **12** are connected by a host bus interface **13**. A memory **14**, a MAC (Multiply Accumulate) unit **15**, a MAC controller **16**, a register set **17** for the host processor **11**, and a register set **18** for the sub-processor **12** are provided for the sub-processor **12**. The MAC unit **15** of the sub-processor **12** is used to perform an MAC operation, and the MAC controller **16** is used to control the processing sequence required for this operation. It should be noted that both the register set **18** for the sub-processor **12** and the register set **17** for the host processor **11** are prepared in the sub-processor **12**.

Programs and data used independently by the host processor **11** are stored in the memory **14**. The sub-processor **12** performs an allocated task, separately from the operation of the host processor **11**, but for the efficient data transfer the host processor **11** and the sub-processor **12** share one part or all of the memory **14**.

The host bus interface **13** is provided to control the exchange of signals between the host processor **11** and the sub-processor **12**. In other words, the interface **13** monitors the signals to ascertain whether the host processor **11** and the sub-processor **12** access each other simultaneously and whether bus contention (conflict state) occurs. Specifically, the interface **13** controls the addresses for the writing and reading of data; transmits an interrupt request (INT) to relay to the host processor **11** a service request (e.g., notification forwarded to the host indicating that a predetermined operation has been completed) from the sub-processor **12**; and monitors the interrupt state.

The MAC operation performed by the MAC unit **15** is an operation whereby a multiplicand (Y) is multiplied by a multiplier (X) and a product is added to a previous result (Prev SUM). In digital signal processing, a matrix operation is performed by repeating the MAC operation. This operation is employed as an application for servo control and so on. Hardware components required for the MAC operation are registers **A1** and **B1** for the storage of the multiplier (X) and the multiplicand (Y), a multiplication unit (MULTIPLIER); an addition unit (ACCUMULATOR); a register (**C1**/**C2**) for the storage of operation results; and a register (**D1**) for the storage of a status. These components are employed to execute the following equation:

$$SUM(i)=X(i)*Y(i)+SUM(i-1)i=1, 2, \ldots \quad \text{[Equation 1]}$$

Assuming that the lengths of the multiplier (X) and of the multiplicand (Y) are each 16 bits, the result (SUM) will be 32 bits, which corresponds to a length that is twice as long. When there is an overflow (either positive or negative) in the result, it is recorded in the register **D1** in which status is recorded.

In order that the MAC unit **15** constituted by the multiplier and the accumulator can be used not only by the sub-processor **12** but also by the host processor **11**, in the sub-processor **12** are provided two types of register sets, which is the feature of the circuit arrangement for this embodiment. That is, not only is the register set **18** provided for the MAC operation performed by the sub-processor **12**, but also the register set **17** for the MAC operation performed at the request of the host processor **11**. The register sets **17** and **18**, for the respective host processor **11** and the sub-processor **12**, each have an **A1** register for storing a multiplier (X), a **B1** register for storing a multiplicand (Y), a **C1**/**C2** register for storing the operation result, and a **D1** register for storing the status. One of these registers is selected in response to a signal obtained by dividing an internal clock, and the MAC operation is performed in a time sharing manner.

With this control method, whereas to the host processor **11** it appears that automatic performance of an MAC operation results from the writing of data to the external register sets, in this embodiment, the MAC operation is performed in a time sharing manner, and no interrupt request need be transmitted to the sub-processor **12**. To acquire the results of an MAC operation, after the host processor **11** writes data required for the MAC operation into the **A1** register and the **B1** register, it then sequentially reads the operation results from the **C1**/**C2** register. In other words, the results of an MAC operation can be obtained merely by writing to and reading from the external register sets.

FIG. **3** is a block diagram illustrating the MAC unit **15** in this embodiment. The MAC unit **15** is constituted by a first selector **21**, a multiplier **22**, an accumulator **23** and a second selector **24**. Upon receipt of a select signal (MAC Sharing), the first selector **21** and the second selector **24** select either the register set **17** for the host processor **11** or the register set **18** for the sub-processor **12**. Specifically, in accordance with the select signal (MAC Sharing), the selector **21**, which has two selectors MUX, supplies to the multiplier **22** the multiplier (X) and the multiplicand (Y) stored in the registers **A1** and **B1**. The multiplier **22** and the accumulator **23** then perform the above described MAC operation under the control of the MAC controller **16**. Thereafter, in response to the select signal (MAC sharing), the second selector **24** supplies the results and the status to the selected **C1**/**C2** and **D1** registers. For an MAC operation that is requested by the host processor **11**, the results and the status are transferred to the host processor **11** via the host bus interface **13**.

The select signal (MAC Sharing) selects the performance of the MAC operation for either the host processor **11** or the sub-processor **12**. A signal obtained by dividing the internal clock of the host processor **11** is employed as the select signal. The divider for the internal clock is determined by the relationship of the clock frequencies for the host processor **11** to those of the sub-processor **12**. Since, as an example, it is possible to perform one MAC operation each clock cycle for a low end application where the internal clock is about 50 MHZ, the assumption that the internal clock is divided by 2 is applied.

When the internal clock is about 50 MHZ, one operation each clock is possible when a 32 bit output (16+16 bits) is implemented for the MAC unit **15**. Thus, for each clock cycle, the state of occupancy of the MAC unit **15** can be switched to accommodate either the host processor **11** or the sub-processor **12**. Since a CISC microcontroller generally needs several clock cycles to write data to an external memory/register, only the writing and reading operation need be performed to obtain the results of the MAC operation, with the processing performed by the sub-processor being little affected.

FIG. **4** is a timing chart for the state when the switching of the following MAC operation is performed in response to the receipt of a ½ frequency division signal.

SUM=*X***1****Y***1**+X***2****Y***2**+(Previous SUM) [Equation 2]

Since this timing is employed for the control of the MAC unit **15**, the MAC unit **15** can be used in a time sharing manner by the host processor **11** and the sub-processor **12**.

In FIG. **4** signal WR represents the writing (Write) to the **A1** register and the **B1** register, and signal RD represents the reading (ReaD) from the **C1**/**C2** and the **D1** registers. Signal HOST Reg **A1** WR (**X1**) represents the writing of the multiplier (X) to the host **A1** register. A multiplicand (**Y1**), a multiplier (**X2**), and a multiplicand (**Y2**) are written in the host processor register. Then, the results (SUM) of the operation **X1*****Y1**+**X2*****Y2** are automatically set in the **C1**/**C2** register, and when an overflow occurs, an overflow flag (STATUS) is set in the **D1** register. Therefore, through the sequential reading of data from the **C1**/**C2** and the **D1** registers, the results of the operation can be obtained. Since several clock cycles are required for the writing and reading by the host processor, only the host processor need write data to the external memory and read required data for the automatic performance of the MAC operation. Since the reading overhead becomes relatively smaller as the number of MAC stages increases, the operational efficiency is improved. The same process can be performed at the same time by the sub-processor.

Even if the writing of the multiplier to the **A1** register is initiated when the select signal (MAC Sharing) selects the sub-processor, the same operation can be performed even though the operation start time is delayed one clock cycle. FIG. **5** is a timing chart for the state when the operation is initiated following one clock cycle delay. "HOST Reg **A1** Data,""HOST Reg **B1** Data" and "**C1**/**C2**/**D2** Data" represent changes in the contents of the individual registers.

The select signal (MAC Sharing) merely determines the timing for the sharing of the MAC unit **15** by the host processor **11** and the sub-processor **12**, and the MAC unit **15** is not always employed in either phase. That is, the select signal merely determines whether the host register set **17** or the sub register set **18** will be connected to the MAC unit **15**.

When the host processor **11** does not require an MAC operation, it is preferable that the select signal (MAC Sharing) constantly select the sub-processor **12**. Therefore, except for a period extending from the time the host processor **11** writes the multiplier (X) to the **A1** register in the register set **17** to the time it reads the status from the **D1** register, the select signal is set to the sub-processor **12**. To do this, the order in which the multiplier (X) and the multiplicand (Y) are written and the order in which signals SUM/L, SUM/H and STATUS are read are specified. When a circuit whose latch is set upon the writing of the first multiplier (X) and is reset by reading signal STATUS, is additionally provided and a frequency divisional signal is gated by the circuit, it is employed as a select signal (MAC Sharing).

The following MAC operation time will now be discussed while employing CISC microcontroller **80C196** as an example.

SUM=*X***1****Y***1**+X***2****Y***2**+(Previous SUM)
with Overflow Check [Equation 3]

At least 90 clock cycles are required to perform this processing for which a common instruction is used. When the MAC unit proposed above is employed, only 56 clock cycles (including the reading of a flag) are required, and the performance can be improved by 38%. Further, while at least 135 clock cycles are required for a three-stage MAC operation, in this case only 72 clock cycles are needed, and the performance can be improved by 47%. As the number of stages for the MAC operation is increased, the degree to which the performance is improved is also increased. The degree of improvement for the performance is increased even more when overflow processing is required.

As is described above, the MAC unit included in the sub-processor is used in a time sharing manner as an external MAC unit for the host processor. Since the MAC unit is employed in a time sharing manner, the period of time required for the operation of the host processor can be reduced, while the operation of the sub-processor is little affected. As a result, the overall performance of the integrated dual processor module can be enhanced.

While the invention has been particularly shown and described with respect to preferred embodiments thereof, it will be understood by those skilled in the art that the foregoing and other changes in form and details may be made therein without departing from the spirit and scope of the invention.

Having thus described our invention, what we claim as new, and desire to secure by Letters Patent is:

1. A shared multiply accumulate (MAC) system comprising:

a host processor;

a sub-processor including multiply accumulate (MAC) unit;

a host bus interface forming a connection between said host processor and said sub-processor;

a first register set in which a multiplier, a multiplicand, a product and a status are written in order for the performance of a multiply accumulate (MAC) operation required by said host processor;

a second register set in which a multiplier, a multiplicand, a product and a status are written in order for the performance of a multiply accumulate (MAC) operation required by said sub-processor; and

selection means for selecting said first register set or said second register set upon receipt of a frequency division signal obtained by the division of an internal clock in said shared MAC system, wherein said MAC unit in said sub-processor performs said MAC operation in accordance with the contents of said first register set or of said second register set that is selected.

2. The shared MAC system according to claim **1**, wherein said selection means includes:

first selection means for, in response to said frequency division signal, supplying to said MAC unit a multiplier and a multiplicand written either in said first register set or in said second register set; and

second selection means for, in response to said frequency division signal, supplying either to said first register set or to said second register set a product obtained by said MAC unit and a status.

3. The shared MAC system according to claim **2**, wherein said MAC unit is constituted by a multiplication unit and an addition unit.

4. The shared MAC system according to claim **2**, wherein said first and said second register sets are provided in said sub-processor.

5. The shared MAC system according to claim **1**, wherein said MAC unit is constituted by a multiplication unit and an addition unit.

6. The shared MAC system according to claim **1**, wherein said first and said second register sets are provided in said sub-processor.

7. A multiply accumulate (MAC) method for a shared MAC system that includes a host processor, a sub-processor having a multiply accumulate (MAC) unit, a host bus interface connected between said host processor and said sub-processor, and a register set, said MAC method comprising the steps of:

writing a multiplier and a multiplicand to said register set via said host bus interface so that a multiply accumulate (MAC) operation required by said host processor can be performed;

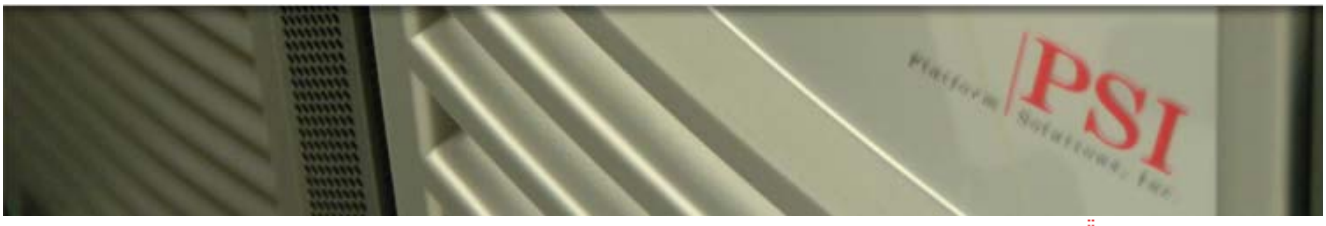
permitting said MAC unit, in response to a frequency division signal obtained by dividing an internal clock in said shared MAC system, to perform said MAC operation in a time sharing manner based on said multiplier and said multiplicand written in said register set;

writing in said register set a product and a status obtained as a result of said MAC operation; and

transmitting said product and said status via said host bus interface to said host processor.

* * * * *

Contact : Support



Home
Products
News
Events
Literature
About
Contact
Board of Directors
Executive Management
Investors
Advisory Committee
Company History
Careers

Newsletter Signup

Contact Sales

408-720-3500

Email Us

About

Platform Solutions, Inc. (PSI), is the first developer of a new generation of mainframe computers compatible with the broadest set of datacenter environments and operating systems, including IBM® z/OS®.

With a vision of offering true choice in the selection of advanced new mainframe systems, PSI provides customers a new level of control over their IT investments and ultimate flexibility in aligning their existing and future enterprise computing resources with changing business priorities and technologies.

Everyday, datacenters balance support for legacy platforms against the need to make strategic investments in new technologies that provide increased business agility and cost efficiency. Resources are aligned to meet service level agreements; and, system integration projects are weighed against complex and time consuming migration efforts. Now, PSI offers these customers a new choice that significantly reduces risk and provides investment protection: a new generation mainframe computer designed specifically to meet the most pressing needs of today's heterogeneous enterprise datacenters.

The new PSI mainframe computers are based on proven systems architecture spun-off from Amdahl and industry-standard Intel® Itanium® 2 processor technology. This unique combination gives customers of PSI systems unequalled system and software cost advantages over proprietary mainframe computers. PSI systems also are the first new generation mainframe computers that can simultaneously run the z/OS® operating system along with Linux, UNIX® and Windows®. This broad compatibility, along with breakthroughs in I/O virtualization, makes the new PSI mainframes ideal for bridging the proprietary mainframe world and today's increasingly open systems based environments.

Originally founded in 1999 by a core team of former Amdahl engineers, PSI raised its first round of venture funding in the Fall of 2003. PSI believes the continued growth in mainframe computing, the rise in more powerful open systems technologies and the company's unequalled insight and expertise in mainframe compatibility provides PSI the rare opportunity to meet customers' need for true choice and flexibility in mainframe computing.

News

The new PSI mainframe computers are based on proven systems architecture spun-off from Amdahl and industry-standard Intel Itanium 2 processor technology.

SUSMAN GODFREY L.L.P.
A REGISTERED LIMITED LIABILITY PARTNERSHIP
ATTORNEYS AT LAW
5TH FLOOR
654 MADISON AVENUE
NEW YORK, NEW YORK 10065
WWW.SUSMANGODFREY.COM

SUITE 5100
901 MAIN STREET
DALLAS, TEXAS 75202-3775
(214) 754-1900

SUITE 950
1901 AVENUE OF THE STARS
LOS ANGELES, CALIFORNIA 90067-6029
(310) 789-3100

SUITE 3800
1201 THIRD AVENUE
SEATTLE, WASHINGTON 98101-3000
(206) 516-3880

SUITE 5100
1000 LOUISIANA STREET
HOUSTON, TEXAS 77002-5096
(713) 651-9366

TIBOR L. NAGY JR.
DIRECT DIAL (212) 336-8332

DIRECT DIAL FAX (212) 336-8340
E-MAIL TNAGY@SUSMANGODFREY.COM

By E-mail June 25, 2008

Edward John Defranco
Quinn Emanuel Urquhart Oliver & Hedges LLP
51 Madison Avenue , 22nd Flr
New York, NY 10010
eddefranco@quinnemanuel.com

Re: *IBM v. PSI*, 06-cv-13565 / Claim Construction

Dear Ed:

In an effort to narrow the issues that need to be addressed by the Court, please note the following:

1. For the "means…for translating said guest logical address…" (claim 9) in the '520 patent, we agree to accept the language you cite in columns 13, 14 and 15 as corresponding structure.
2. We agree to accept your construction for "floating point unit having an internal dataflow" (claim 11) in the '106 patent, in view of the fact that the claim expressly requires "an internal floating point format." Also, as noted in our brief, we have agreed to accept "operations" instead of "calculations" for "floating point unit," so that our construction for "floating point unit" is now: That portion of a processor's circuitry that performs floating point calculations.
3. For target routine ('261 patent) we withdraw the second part of our construction, so that we now have the following shortened construction for target routine: a defined sequence of target instructions. This should simplify the analysis for you and the Court.

In addition, we have the following two proposals:

1. We propose the following compromise construction for "processor": One or more integrated circuits, including any microcode embedded in the circuits, that decodes and executes instructions.

2. We will accept your construction for "instruction set" ('520 patent) if you agree not to argue that a computer system can have *two or more* instruction set architectures.

Please let me know if you agree to these proposals.

Sincerely,

Tibor L. Nagy, Jr.
Attorney for PSI

PROPOSED CONSTRUCTIONS

Terms Common to Several Patents	IBM Proposed Construction	PSI's Proposed Construction
processor (‘495, ‘789, ‘520, ‘261, and ‘709 patents)	A portion of a computer system that interprets and executes instructions.	one or more integrated circuits that interprets (i.e., decodes) and executes instructions.
instruction (‘495, ‘789, and ‘709 patents)	A language construct, which can be a string of digits, that specifies an operation and identifies its operands, if any.	A string of digits that specifies an operation and identifies its operands, if any, and can be directly executed by the processor to which it is directed.
program status word (‘495 and ‘678 patents)	A defined set of data that indicates the next instruction to be executed and includes the program condition code and program authority, where that data directs the processor in the execution of a program.	The contents of the register that indicates the next instruction to be executed, includes the program condition code and program authority, and directs the processor in the execution of a program.
register (‘495 and ‘678 patents)	A part of internal storage having a specified storage capacity and usually intended for a specific purpose.	A hardware storage element in the processor that can be accessed faster than memory.

U.S. Patent No. 5,987,495

'495 Claim Term	IBM Proposed Construction	PSI's Proposed Construction
means for decoding a program instruction specifying a register selected from said set of registers, said register pointing to a save area containing a saved program status word and saved register contents	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* Decoding a program instruction specifying a register selected from the set of registers, the register pointing to a save area containing a saved program status word and saved register contents *Structure:* A processor (*see, e.g.*, Fig. 1, 102; 5:4-11; or Col. 7:21-28) configured to decode an instruction from a program executing in said problem state specifying a storage location containing a saved program status word (*see, e.g.*, Figs. 2A, 2B, 8:63-65, 9:5-15) and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* decoding a program instruction specifying a register selected from said set of registers, said register pointing to a save area containing a saved program status word and saved register contents *Structure:* No corresponding structure.
means response to said decoding means for executing said instruction, said executing means comprising:	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* Executing the instruction *Structure:* A processor configured to execute an instruction (*see, e.g.*, Fig. 1, 102; Col. 5:4-11; Col. 6:66 – 7:7, or Col. 7:21-28), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* executing said instruction *Structure:* No corresponding structure.

'495 Claim Term	IBM Proposed Construction	PSI's Proposed Construction
means for accessing said save area using the contents of the register specified by said program instruction	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* Accessing the save area using the contents of the register specified by the program instruction *Structure:* A processor (see, e.g., Fig. 1; 102; Col. 7:21-28; or 5:4-11) that is configured to access the save area using the contents of the register specified by the program instruction (*see*, *e.g.*, Fig. 2A; Fig. 2B, Fig. 3A; Fig. 6, steps 601, 602, 604, 606; Col. 8:63-65; Col. 9:5-22; Col. 10:42-45; Col. 10:46-50; Col. 10:54-59; or Col. 10:62-66), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* accessing said save area using the contents of the register specified by said program instruction *Structure:* No corresponding structure.

'495 Claim Term	IBM Proposed Construction	PSI's Proposed Construction
means for restoring said program status word and said register from the saved program status word and saved register contents contained in said save area to resume execution at the instruction address contained in said saved program status word with the program context defined by said saved program status word and saved register contents.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* Restoring the program status word and the register from the saved program status word and saved register contents contained in the save area to resume execution at the instruction address contained in the saved program status word with the program context defined by said saved program status word and saved register contents. *Structure:* A processor (*see, e.g.*, Fig. 1; 102; Col. 7:21-28; or 5:4-11) that is configured to restore the program status word and the register from the saved program status word and saved register contents contained in the same area (*see, e.g.*, Fig. 8, step 810; Fig. 6, steps 603, 605, or 607; Col. 9:58-64; Col. 10:50-53; Col. 10:59-61; or Col. 10:66-11:2), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* restoring said program status word and said register from the saved program status word and saved register contents contained in said save area to resume execution at the instruction address contained in said saved program status word with the program context defined by said saved program status word and saved register contents. *Structure:* Fig. 6.

U.S. Patent No. 6,775,789

'789 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
usable as a current time of day clock value in real-time processing	The sequence value reflects the actual time at which the time value was requested, and is useable as a sequential clock value where later requests always result in larger values	Representing the actual time of day at which the value can be used by a program
at least one computer usable medium having computer readable program code means embodied therein for causing the generating of unique sequence values usable within a computing environment, the computer readable program code means in said article of manufacture comprising:	Governed by 35 U.S.C. § 112, ¶ 6. *Function*: causing the generating of unique sequence values usable within a computing environment. *Structure*: An article of manufacture (*see, e.g.*, Col. 14:61-15:4) having computer usable media for causing the generating of unique sequence values usable within a computing environment (*see, e.g.*, Fig.9; col. 8:62-67; Fig. 10, step 1010, 1002; col. 9:1-5; col. 9:13-15; col. 11:23-35; or col. 11:61-12:6.), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* causing the generating of unique sequence values usable within a computing environment *Structure:* No corresponding structure.

'789 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
computer readable program means for causing a computer to provide as one part of a sequence value timing information comprising at least one of time-of-day information and date information;	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* causing a computer to provide as one part of a sequence value timing information comprising at least one of time-of-day information and date information *Structure:* An article of manufacture (*see, e.g.*, col. 14:61-15:4) having computer usable media for causing a computer to provide as one part of a sequence value timing information comprising at least one of time-of-day information and date information (*see, e.g.*, Fig. 9; Col. 8:62-67; Fig. 10, step 1002; or Col. 9:1-5), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* causing a computer to provide as one part of a sequence value timing information comprising at least one of time-of-day information and date information *Structure:* No corresponding structure.

'789 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
computer readable program means for causing a computer to include as another part of said sequence value selected information usable in making said sequence value unique across a plurality of operating system images on one or more processors of said computing environment, wherein said sequence value is usable as a current time of day clock value in real-time processing by one or more processors of the computing environment.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* causing a computer to include as another part of the sequence value selected information usable in making the sequence value unique across a plurality of operating system images on one or more processors of the computing environment *Structure:* An article of manufacture (*see, e.g.*, Col. 14:61-15:4) having computer usable media for causing a computer to include as another part of said sequence value selected information usable in making said sequence value unique across a plurality of operation system images on one or more processors of said computing environment (*see, e.g.*, Fig. 9; Col. 8:62-67; Fig. 10, step 1010; Col. 9:13-15; Col. 11:23-35; or Col. 11:61-12:6), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* causing a computer to include as another part of the sequence value selected information usable in making the sequence value unique across a plurality of operating system images on one or more processors of the computing environment, wherein said sequence value is usable as a current time of day clock value in real-time processing by one or more processors of the computing environment *Structure:* No corresponding structure.

U.S. Patent No. 5,953,520

'520 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
instruction set	The complete set of the operations of the instructions of a computer architecture together with the types of meanings that can be attributed to their operands.	the complete set of the operations of the instructions of a computer or a computer architecture together with a description of the types of meanings that can be attributed to their operands
semantic routine	A defined set or sequence of native instructions that, when executed, emulates an associated guest instruction.	a defined sequence of native instructions that, when executed, emulates an associated guest instruction
means, responsive to receipt of said guest memory access instruction for emulation, for translating said guest logical address into a guest real address and for thereafter translating said guest real address into a native physical address	Governed by 35 U.S.C. § 112, ¶ 6 *Function:* translating said guest logical address into a guest real address and for thereafter translating said guest real address into a native physical address *Structure:* A data processing system (*see*, *e.g.*, Figs. 1, 2, or 3) configured to translate the guest logical address into a guest real address and thereafter to translate the guest real address into a native physical address (*see*, *e.g.*, Figs. 7, 8, 9; column and lines: 13:52-14:45; or 14:46-15:7; or 15:56-65), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* translating said guest logical address into a guest real address and for thereafter translating said guest real address into a native physical address *Structure:* Fig. 7, 8, 9; and column and lines 13:52-14:45; or 14:46-15:7; or 15:56-65.

'520 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
means for executing a semantic routine that emulates said guest memory access instruction utilizing said native physical address.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* executing a semantic routine that emulates said guest memory access instruction utilizing said native physical address *Structure:* A data processing system (*see, e.g.*, Figs. 1, 2, or 3) configured to execute a semantic routine that emulates the guest memory access instruction utilizing the native physical address (*see, e.g.*, Figs. 8, 9; column and lines: 11:4-15; 11:26-31; 13:60-14:25; 15:56-67), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* executing a semantic routine that emulates said guest memory access instruction utilizing said native physical address *Structure:* Fig. 7, 8, 9

U.S. Patent No. 6,009,261

'261 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
patching instruction	An element of a target routine that is used to copy or modify some or all of a target instruction, to enable a guest instruction to be emulated.	an element of a target routine that is used to copy or modify some or all of a target instruction, to enable a guest instruction to be emulated in a dynamic manner each time a guest instruction is encountered
incompatible instruction	A language, which can be a string of digits, construct that specifies an operation and identifies its operands, if any, and pertains to the architecture being emulated, which is different than the architecture on which the emulator runs.	A string of digits that specifies an operation and identifies its operands, if any, and would be able to be directly executed by a processor of the emulated data processing system.
target instruction	A language construct, which can be a string of digits, that specifies an operation and identifies its operands, if any, and pertains to the architecture on which the emulator runs.	A string of digits that specifies an operation and identifies its operands, if any, and can be directly executed by the processor to which it is directed.
target routine	A set or sequence of target instructions.	a defined sequence of target instructions.

U.S. Patent No. 5,825,678

'678 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
a floating point processor	A portion of a computer system that interprets and executes floating point instructions.	A processor that contains a floating point unit. A floating point unit is that portion of a processor's circuitry that executes floating point instructions by performing operations on floating point numbers.
a machine instruction	A string of digits that specifies an operation and identifies its operands, if any, and can be directly executed by a processor of a computer.	A string of digits that specifies an operation and identifies its operands, if any, and can be directly executed by the processor to which it is directed.
means for retrieving the floating point number from memory	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* retrieving the floating point number from memory *Structure:* A shared memory computer system (*see, e.g.*, Fig. 1, 2:35-38; Fig. 2, 2:39-59) for retrieving the floating point number from memory (*see, e.g.*, Fig. 8, 3:63-66), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* retrieving the floating point number from memory *Structure:* No corresponding structure.

'678 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
means for determining whether the data class of the floating point number is the identified data class by examination of condition of the fields of the floating point number	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* determining whether the data class of the floating point number is the identified data class by examination of condition of the fields of the floating point number *Structure:* A shared memory computer system (*see, e.g.*, Fig. 1, 2:35-38; Fig. 2, 2:39-59) for determining whether the data class of the floating point number is the identified data class by examination of condition of the fields of the floating point number (*see, e.g.*, Fig. 5, 3:34-45; Fig. 7, 3:49-50; Fig. 8, 3:66-4:5; Fig. 9, 4:9-32), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* determining whether the data class of the floating point number is the identified data class by examination of condition of the fields of the floating point number. *Structure:* Fig. 9, except for the circuitry that connects AND gates to the box marked CC in Fig. 9 (known in the art as a "wired OR gate").
means for setting a condition code in a program status word based upon the determination of whether the data class is the identified data class	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* setting a condition code in a program status word based upon the determination of whether the data class is the identified data class *Structure:* A shared memory computer system (*see, e.g.*, Fig. 1, 2:35-38; Fig. 2, 2:39-59) for setting a condition code in a program status word based upon whether the data class is the identified data class (*see, e.g.*, Fig. 8, 4:5-8; Fig. 9, 4:9-32), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* setting a condition code in a program status word based upon the determination of whether the data class is the identified data class *Structure:* The circuitry that connects the AND gates to the box marked CC in Fig. 9 (known in the art as a "wired OR gate").

U.S. Patent No. 5,687,106

'106 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
floating point unit	Part of a processor that performs floating point operations.	That portion of a processor's circuitry that performs floating point operations
supports both said first floating point architecture and said second floating point architecture	Provides the necessary resources for the correct operation of both the first and second floating point architectures.	A floating point unit "supports" a floating point architecture when it can, on its own, perform calculations on floating point numbers of that architecture.
converter	Hardware, or a combination of hardware and software, that changes data from one format or architecture type to another format or architecture type.	Circuitry within the floating point unit that changes the format or architecture type of a floating point number.

U.S. Patent No. 6,654,812

'812 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
host-network interface	A communication interface coupled between a mainframe class data processing system and a port to a network.	A hardware component coupled between a mainframe class data processing system and a network port.
saving at said host-network interface	Saving at said host-network interface; or, saving on said host-network interface..	saving on memory located within the host-network interface
(computer) readable program code means embodied therein for causing network communications in a mainframe class data processing system having multiple partitions and a port to a network, the computer readable program code means in the article of manufacture comprising:	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* causing network communications in a mainframe class data processing system having multiple partitions and a port to a network *Structure:* An article of manufacture (*see, e.g.*, 10:46-57 and equivalents thereof) that causes a computer to cause network communications in a mainframe class data processing system having multiple partitions and a port to a network (*see, e.g.*, Figs. 5, 6, 7, 8A, 8B; or col. 7:45-8:19; 8:31-48; 9:8-10; 9:8-10; 9:46-57; 10:1-11), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* causing network communications in a mainframe class data processing system having multiple partitions and a port to a network *Structure:* No corresponding structure.

'812 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
(i) computer readable program code means for causing a computer to effect saving at a host-network interface an internet protocol (IP) address of at least one of the multiple partitions of the mainframe class data processing system	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* causing a computer to effect saving at a host-network interface an internet protocol (IP) address of at least one of the multiple partitions of the mainframe class data processing system *Structure:* An article of manufacture (*see, e.g.*, 10:46-57) that causes a computer to effect saving at a host-network interface an internet protocol (IP) address of at least one of the multiple partitions of the mainframe class data processing system (*see, e.g.*, Fig. 5, 7:66-8:4), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* causing a computer to effect saving at a host-network interface an internet protocol (IP) address of at least one of the multiple partitions of the mainframe class data processing system *Structure:* No corresponding structure.

'812 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
(ii) computer readable program code means for causing a computer to effect generating an IP datagram at a first partition of said multiple partitions to be forwarded to a second partition of said multiple partitions using a destination IP address	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* causing a computer to effect generating an IP datagram at a first partition of said multiple partitions to be forwarded to a second partition of said multiple partitions using a destination IP address *Structure:* An article of manufacture (*see, e.g.*, 10:46-57) that causes a computer to effect generating an IP datagram at a first partition of said multiple partitions to be forwarded to a second partition of said multiple partitions using a destination IP address (*see, e.g.*, step 200 of Fig. 6, 8:33-37; step 300 of Fig. 7, 9:8-10; Fig. 8A, 9:46-57; or Fig. 8B, 10:1-11), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* causing a computer to effect generating an IP datagram at a first partition of said multiple partitions to be forwarded to a second partition of said multiple partitions using a destination IP address *Structure:* No corresponding structure.

'812 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
(iii) computer readable program code means for causing a computer to effect determining whether said destination IP address for said IP datagram comprises an IP address saved at said host-network interface for said at least one partition, and if so, forwarding the IP datagram directly from said first partition to said second partition of said multiple partitions without employing said network.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* causing a computer to effect determining whether the destination IP address for the IP datagram comprises an IP address saved at the host-network interface for said at least one partition, and if so, forwarding the IP datagram directly from the first partition to the second partition of said multiple partitions without employing the network. *Structure:* An article of manufacture (*see, e.g.*, 10:46-57) that causes a computer to effect determining whether the destination IP address for the IP datagram comprises an IP address saved at the host-network interface for said at least one partition, and if so, forwarding the IP datagram directly from the first partition to the second partition of said multiple partitions without employing the network (*see, e.g.*, Fig. 5, 8:20-30; or steps 210-240 of Fig. 6, 8:37-48), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* causing a computer to effect determining whether the destination IP address for the IP datagram comprises an IP address saved at the host-network interface for said at least one partition, and if so, forwarding the IP datagram directly from the first partition to the second partition of said multiple partitions without employing the network. *Structure:* No corresponding structure.

U.S. Patent No. 6,971,002

'002 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
firmware	The term "firmware" is only used as a modifier in the phrase "firmware image" and does not need to be construed separately.	Firmware is "software" stored in a memory chip that holds its content without electrical power, such as, for example, read-only memory (ROM), programmable ROM (PROM), erasable programmable ROM (EPROM), electrically erasable programmable ROM (EEPROM), and non-volatile random access memory (non-volatile RAM).
firmware image	Software copied from non-volatile memory and used to boot a partition.	An instance of a particular firmware.
storing a plurality of different firmware images in said computer system / a plurality of different firmware images being stored in said computer system	storing at least two firmware images that are not the same as each other	storing at least two firmware images that are not the same as each other
capable of being executed during a power-on process to boot said computer system	Capable of being executed during the operations performed by a computer system from the time it is turned on until and including executing the partition firmware to boot said computer system.	able to be executed during the operations performed by a computer system from the time it is turned on until it begins executing the partition firmware

'002 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
instruction means for storing a plurality of different firmware images in said computer system	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* instructing a computer to store a plurality of different firmware images in the computer system *Structure:* A computer program product stored in a computer recordable-type media (*see, e.g.*, 8:4-20) that instructs a computer to store a plurality of different firmware images in the computer system (*see, e.g.*, Fig. 3, 6:13-16), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* storing a plurality of different firmware images in said computer system *Structure:* No corresponding structure
instruction means for rebooting one of said plurality of partitions utilizing one of said plurality of firmware images without rebooting other ones of said plurality of partitions	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* instructing a computer to reboot one of the plurality of partitions utilizing one of the plurality of firmware images without rebooting other partitions *Structure:* A computer program product stored in a computer recordable-type media (*see, e.g.*, 8:4-20) that instructs a computer to reboot one of the plurality of partitions utilizing one of the plurality of firmware images without rebooting other partitions (*see, e.g.*, Fig. 3, 6:58-67; or blocks 422-430 of Fig. 4, 7:30-45), and equivalents thereof.	Governed by 35 U.S.C. § 112, ¶ 6. *Function:* rebooting one of said plurality of partitions utilizing one of said plurality of firmware images without rebooting other ones of said plurality of partitions *Structure:* No corresponding structure

U.S. Patent No. 5,414,851

'851 Claim Term	IBM's Proposed Construction	PSI's Proposed Construction
input/output control blocks	Data structures containing information about I/O resources.	a hardware or microprogramming construct which specifies a shared resource to an OS, and may be said to represent an image of the resource to each sharing OS

Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution

YALE PATT, FELLOW, IEEE

Invited Paper

The first microprocessor, the Intel 4004, showed up in 1971. It contained 2300 transistors and operated at a clock frequency of 108 kHz. Today, 30 years later, the microprocessor contains almost 200 million transistors, operating at a frequency of more than 1 GHz. In five years, those numbers are expected to grow to more than a billion transistors on a single chip, operating at a clock frequency of from 6 to 10 GHz.

The evolution of the microprocessor, from where it started in 1971 to where it is today and where it is likely to be in five years, has come about because of several contributing forces. Our position is that this evolution did not just happen, that each step forward came as a result of one of three things, and always within the context of a computer architect making tradeoffs. The three things are: 1) new requirements; 2) bottlenecks; and 3) good fortune. I call them collectively agents for evolution.

This article attempts to do three things: describe a basic framework for the field of microprocessors, show some of the important developments that have come along in the 30 years since the arrival of the first microprocessor, and finally, suggest some of the new things you can expect to see in a high-performance microprocessor in the next five years.

Keywords*—*Computer architecture, microarchitecture, microprocessor, microprocessor design, microprocessor evolution.

I. Basic Framework

A. Computer Architecture: A Science of Tradeoffs

Computer architecture is far more "art" than "science." Our capabilities and insights improve as we experience more cases. Computer architects draw on their experience with previous designs in making decisions on current projects. If computer architecture is a science at all, it is a science of tradeoffs. Computer architects over the past half century have continued to develop a foundation of knowledge to help them practice their craft. Almost always the job of the computer architect requires using that fundamental knowledge to make tradeoffs. This has been especially true throughout the evolution of the microprocessor.

Manuscript received May 29, 2001; revised August 6, 2001.

The author is with the University of Texas at Austin, Austin, TX 78712-1084 USA (e-mail: patt@ece.utexas.edu).

Publisher Item Identifier S 0018-9219(01)09681-5.

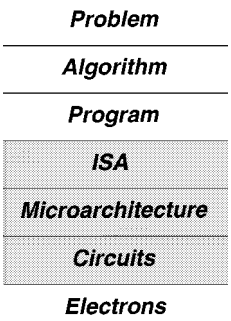


Fig. 1. The microprocessor today.

B. Levels of Transformation

Numbers of transistors and their switching times are resources provided by process technology. What we use those resources for depends on the demands of the marketplace. How we use those resources is what the microprocessor is all about. Fig. 1 shows the levels of transformation that a problem, stated in some natural language like English, must go through to be solved. In a real sense, it is the electrons that actually do the work and solve the problem. However, since we do not speak "electron" and electrons do not speak any natural language, the best we can do is systematically transform the problem through the levels shown on Fig. 1 until we reach the electron (or device) level, that is, the 200 million transistor, 1-GHz chip.

Along the way, the problem solution is first formulated as an algorithm to remove the unacceptable characteristics of natural language, such as ambiguity. It is then encoded in a mechanical language and compiled to the instruction set architecture (ISA) of the particular microprocessor. The ISA is the agreed upon interface that: 1) the compiled program uses to tell the microprocessor what it (the program) needs done and 2) the microprocessor uses to know what it must carry out

in behalf of the program. The ISA is implemented by a set of hardware structures collectively referred to as the microprocessor's microarchitecture. Each hardware structure and its interconnections are made of electronic digital circuits, which in turn are made of electronic devices.

When we say "microprocessor" today, we generally mean the shaded region of Fig. 1. That is, each microprocessor consists of circuits which implement hardware structures (collectively called the microarchitecture) which provide an interface (called the ISA) to the software. In the case of the personal computer, the ISA is the IA-32, and the microarchitecture is Intel's Pentium IV, or in earlier days, the Pentium III, Pentium II, Pentium Pro, 486, etc., or AMD's K-8, or in earlier days, K-7, K-6, etc.

There are other ISAs; for example, SPARC (from Sun Microsystems), Alpha (from Compaq), and Power-PC (from Motorola and IBM). Each has its own idiosyncrasies that makes it a better or worse interface for what a compiler can deliver, or how the microarchitecture can carry out the work. For each ISA, there are multiple distinct microarchitectures. We have mentioned several for the IA-32. For the Alpha, there are, for example, the 21064, 21164, and 21 264.

At each step in the hierarchy, from choice of algorithm, to language, to ISA to microarchitecture, to circuits, there are choices and therefore tradeoffs.

Often, but not always, the choice is between higher performance and lower cost. An analogy to the automobile is instructive. One can build a high-performance sports car that can go from 0 to 100 mph in practically 0 seconds. But, it will be very expensive. Or, one can build a very inexpensive automobile that could never get to 100 mph, but gets 100 miles to a gallon of gasoline. One does not get performance and economy. That is the tradeoff.

C. Design Points

The design of a microprocessor is about making relevant tradeoffs. We refer to the set of considerations, along with the relevant importance of each, as the "design point" for the microprocessor—that is, the characteristics that are most important to the use of the microprocessor, such that one is willing to be less concerned about other characteristics. Performance, cost, heat dissipation, and power consumption are examples of characteristics that strongly affect a design point. Another is "high availability"—one can design a microprocessor where the most important consideration is the requirement that the microprocessor never fail. Some customers are willing to accept lower performance or higher cost if they can be assured that the microprocessor will never fail. We call such a processor "fault tolerant," or highly available.

Other customers are willing to sacrifice a little performance, if it is combined with substantial savings in energy requirements. This design point has become more and more important as the power and energy requirements of the highest performance chips have become unacceptably larger and larger. Again, there is a tradeoff: highest performance or power awareness.

It is worth noting that "power awareness" is different from another important design point, "low power." There are many applications where the overriding consideration is that the microprocessor operate for a long period of time using a very small energy source.

In each case, it is usually the problem we are addressing (see again Fig. 1) which dictates the design point for the microprocessor, and the resulting tradeoffs that must be made.

D. Application Space

The word "Problem" in Fig. 1 is a catch-all for the Application Space, that is, the set of applications for which we wish to use microprocessors. This set is increasing at a phenomenal rate, and is expected to continue to do so. In fact, as long as people dream up more uses for computers, the need for microprocessors and the tradeoffs that each will make will continue to expand. That is, the application space (or, rather, the applications of central importance) drive the design point. We have already mentioned high-availability processors where the applications demand that the microprocessor never fails. And low power processors where the applications must be able to run for a long time on a small amount of energy.

Other examples of the application space that continue to drive the need for unique design points are the following:

1) scientific applications such as those whose computations control nuclear power plants, determine where to drill for oil, and predict the weather;
2) transaction-based applications such as those that handle ATM transfers and e-commerce business;
3) business data processing applications, such as those that handle inventory control, payrolls, IRS activity, and various personnel record keeping, whether the personnel are employees, students, or voters;
4) network applications, such as high-speed routing of Internet packets, that enable the connection of your home system to take advantage of the Internet;
5) guaranteed delivery (a.k.a. real time) applications that require the result of a computation by a certain critical deadline;
6) embedded applications, where the processor is a component of a larger system that is used to solve the (usually) dedicated application;
7) media applications such as those that decode video and audio files;
8) random software packages that desktop users would like to run on their PCs.

Each of these application areas has a very different set of characteristics. Each application area demands a different set of tradeoffs to be made in specifying the microprocessor to do the job.

E. The Basics of Processing

Simply put, a microprocessor processes instructions. To do this, it has to do three things: 1) supply instructions to the core of the processor where each instruction can do its job; 2)

supply data needed by each instruction; and 3) perform the operations required by each instruction.

F. Instruction Supply

In the early days of supplying instructions, one instruction was fetched at a time, decoded, and sent to the core for processing. As time has passed, the number that can be fetched at one time has grown from one to four, and shows signs of soon growing to six or eight. Three things can get in the way of fully supplying the core with instructions to process: instruction cache misses, fetch breaks, and conditional branch mispredictions. When an access to the instruction cache fails, the supply of instructions drops to zero until the cache miss is serviced. A fetch break occurs when an instruction being fetched is a taken branch, rendering useless all the subsequent instructions fetched in the same cycle, independent of the issue width. A conditional branch misprediction means that all instructions fetched since the mispredicted branch represent wasted effort, and must be thrown away before proceeding along the correct instruction path.

G. Data Supply

To supply data needed by an instruction, one needs the ability to have available an infinite supply of needed data, to supply it in zero time, and at reasonable cost. Real data storage can not accommodate these three requirements. The best we can do is a storage hierarchy, where a small amount of data can be accessed (on-chip) in one to three cycles, a lot more data can be accessed (also, on-chip) in ten to 16 cycles, and still more data can be accessed (off chip) in hundreds of cycles. The result is that real data storage suffers from latency to obtain a particular data element and the bandwidth necessary to move that data element from its location in the storage hierarchy to the core of the processor where it is needed.

As bad as this off-chip latency is today, it is getting worse all the time. Improvements in processor cycle time continue to grow at a much faster rate than memory cycle time. In a few years we expect to see off-chip data accesses to memory take thousands of processor cycles.

H. Instruction Processing

To perform the operations required by these instructions, one needs a sufficient number of functional units to process the data as soon as the data is available, and sufficient interconnections to instantly supply a result produced by one functional unit to the functional unit that needs it as a source. However, sufficient interconnections are not enough. As on-chip cycle times decrease, the latency required to forward results produced by functional units in one part the chip to functional units in other parts of the chip where these results are needed as source operands gets worse.

II. Agents for Evolution

Many things have aided the development of the microprocessor: The willingness of the buying public to scoop up what the vendors produce—without a market, we would have all gone home long ago. The creativity of engineers to come up with answers where there were problems—without solutions, there would be no evolution.

I submit that these things come second, and that the forcing functions (which I have called Agents for Evolution) have been new requirements, bottlenecks, and good fortune.

A. Agent I: New Requirements

Early microprocessors limited processing to what one could achieve by fetching one instruction each cycle, decoding that instruction and forwarding it and its data to the functional units in the core for processing. The demand for higher performance dictated that fetching one instruction each cycle was insufficient. The result is the wide-issue microprocessor, where the fetch mechanism allows multiple instructions to be fetched, decoded and issued to the execution core each cycle.

Another example, also due to requirements for high performance, was the need for more than one instruction to be processed at the same time. One can do only one ADD at a time if one has only one ALU. The result was the inclusion of multiple functional units in the execution core.

Today the prevailing new requirement involves power consumption, or what is being referred to as power-aware computing. The requirement is to provide the same level of computer performance as a previous design, while consuming a fraction of the power required for that previous design. Note that this is different from the low-power requirement of embedded processors, which has been an important design point for some time.

There is a good deal of sentiment that tomorrow's new requirement will involve the human interface, which is demanding more and more attention as computer/human interaction becomes more and more pervasive.

B. Agent II: Bottlenecks

We have identified above the three components of instruction processing (instruction supply, data supply, and carrying out the operations of the instruction), and what each entails. By far, most of the improvements to the microprocessor have come about due to attempts to eliminate bottlenecks that prevent these three components from doing their jobs.

For example, instruction supply requires fetching some number—today four—instructions each cycle. If these instructions were stored in memory, the time to fetch would be too long. The bottleneck is the slow memory. Thus, the instruction cache was invented.

If the hardware has the capability to fetch four instructions, but the second instruction is a conditional branch, only two—not four—instructions would be fetched. The bottleneck, which is caused by the conditional branch, is the arrangement of instructions in the order produced by the compiler, rather than the (dynamic) order in which the instructions are executed. The new feature, first added recently to the Pentium IV, is the Trace Cache, which stores instructions in the order in which they have recently been executed, not in the (static) order designated by the compiler.

Finally, if instructions are to be supplied every cycle, one has a problem when encountering a branch in that the condition that determines whether or not the branch should be taken is not yet known. One could wait for that condition to be resolved, temporarily halting the fetch of instructions until this resolution occurs. That bottleneck was alleviated by the introduction of branch predictors, which guess whether the branch should be taken or not, and immediately fetch according to this guess.

C. Agent III: Good Fortune

Good fortune happens when something causes a windfall which can then be used to provide additional features to the microprocessor. A good example of this is the technology shrink that allows a next implementation of a microprocessor to take up less space on the chip than the previous implementation did. With less space required by the old design, more space is available for doing other things. Two examples of other things that were introduced to the microprocessor in this way were the on-chip floating point accelerator in the mid 1980s and the multimedia instruction extension capability added on-chip in the late 1990s.

III. Evolution: From 1971 to Today

The microprocessor has evolved dramatically from the simple 2300 transistors of the Intel 4004 to what it is today. As suggested above, that evolution was due to several things. The result is that the Pentium IV of today bears little resemblance to the Intel 4004 of 1971.

Some examples of that evolution are the following.

A. Pipelining

Early microprocessors processed one instruction from fetch to retirement before starting on the next instruction. Pipelining, which had been around at least since the 1960s in mainframe computers, was an obvious solution to that performance bottleneck. Commercially viable microprocessors such as the Intel 8086 introduced the first step toward pipelining in the late 1970s by prefetching the next instruction while the current instruction was being executed.

B. On-Chip Caches

On-chip caches did not show up in microprocessors until a few years later. The latency to get instructions and data from off-chip memory to the on-chip processing elements was too long. The result: an on-chip cache. The first commercially viable microprocessor to exhibit an on-chip cache was the Motorola MC68020, in 1984. In a pipelined processor, it is useful to be able to fetch an instruction and fetch data in the same cycle without the bottleneck of contending for the one port to the cache. Once it became possible to put caches on the chip, the next step was to cache separately instructions and data. Among the first microprocessors to include separate on-chip instruction and data caches was Motorola's MC68030, in 1986.

A cache can either be fast or large, not both. Since the cache had to be fast, it had to also be small, resulting in too large a cache miss ratio. The problem with cache misses was the delay to go off-chip to satisfy the miss was too large. The result: two levels of cache on-chip, so that a miss in the fast, small first level cache could be satisfied by a larger, slower second level cache that was still a lot faster than going off-chip. This feature did not show up on microprocessors until the Alpha 21164 around 1994. Today, almost all high-performance microprocessors have two levels of cache.

C. Branch Prediction

The benefits of pipelining are lost if conditional branches produce pipeline stalls waiting for the condition on which the branch is based to be resolved. Hardware (run-time) branch predictors did not show up on the microprocessor chip until the early 1990s. Some of the early microprocessors to introduce run-time branch predictors were Motorola's MC88110, Digital's Alpha 21064, and Intel's Pentium.

D. On-Chip Floating Point Unit

Early microprocessors had a separate chip to handle floating point operations. As transistors got smaller and chips got larger, the transistor count reached the point where the floating point unit could be placed on the same chip with the main processing unit, utilizing "new" spare capacity and saving unnecessary off-chip communication. The Motorola MC88100 and the Intel 486 were two early chips to incorporate the floating point unit on the main processor chip in the late 1980s.

E. Additional Specialized Functional Units

Early microprocessors had one or very few functional units. As the number of transistors on a chip grew, so also the recognition that concurrent execution could be exploited with multiple functional units. First such things as separate address ALUs were added. Then a more sophisticated load/store functional unit containing structures like write buffers, a miss pending queue, and a mechanism for handling memory disambiguation became a part of the general microprocessor chip in the 1990s. Intel's i860, in 1986, was one of the first to have multiple specialized functional units, one to assist graphical processing, in addition to the floating point add and multiply units.

F. Out-of-Order Processing

The contract between the programmer/compiler and the microarchitecture requires that instructions must be carried out in the order specified by the translated program. This produces a bottleneck each time an instruction that can not be carried out prevents a subsequent instruction from being executed if the subsequent instruction has all that it needs to begin execution. The mechanism to get around this bottleneck, out-of-order processing, had been known since the mid-1960s on the IBM 360/91, for example. However, the mechanism was restricted to high-performance scientific computation, where it was claimed that being able to handle precise exceptions was not a critical requirement. Current acceptance of the IEEE Floating Point Standard by just

about all manufacturers suggests otherwise. Nonetheless, although out-of-order execution had been used on mainframes for 35 years, its use in combination with precise exception handling first showed up on microprocessors in the mid-1990s.

To accommodate out-of-order execution, the microprocessor adopted the register aliasing and reservation stations that had been used on earlier mainframes. To do so with precise exceptions, the microprocessor had to add the distinction between instruction execution and instruction retirement. Instructions were allowed to execute whenever their resources (data and functional units) became available, independent of their order in the program, but were forced to retire in the same order that they occurred in the executing program. That is, the internal microarchitecture could execute instructions out of order, but had to report results (i.e., change the permanent state of the computation) in the order the instructions occurred in the executing program. Doing this required a structure for restoring the state in the case of an exception. This state restoring mechanism is commonly manifested as a Reorder Buffer on most microprocessors today, and as a checkpoint retirement structure on a few. Though other microprocessors showed the beginnings of out-of-order execution earlier, the first to fully exploit the concept was the Pentium Pro, in 1995.

G. Clusters

Single die size continues to increase, feature size continues to decrease, and on-chip frequencies continue to increase. The result is that a value produced by a functional unit at one corner of the chip can not traverse the chip and be available as a source to a functional unit at the opposite corner of the chip in the next cycle. The result—partition the execution core into clusters so that most of the time, results produced by a functional unit in one cluster will be used by another functional unit in the same cluster. One still has the problem of knowing which cluster to steer a particular instruction to, but if successful, the inordinate multiple cycle delay caused by a result having to traverse a major part of the chip goes away. This feature first appeared on the Alpha 21264 in the late 1990s.

H. Chip Multiprocessor

An alternative use of the increasing richness of the die (many more transistors, combined with faster operating frequency) is to partition the chip into regions, with an identical processor occupying each region. The paradigm is referred to as CMP, for chip multiprocessor. For tasks that are easily partitionable into self-contained instruction streams, where substantial communication between the instruction streams is required, the CMP is a useful paradigm. It provides the added benefit of interprocessor communication occurring on-chip, where such communication is much faster than off-chip. IBM introduced this feature in 2000, with two processors on its G4 chip.

I. Simultaneous Multithreading

Instruction supply suffers when the instruction cache access results in a cache miss. A lot of capacity is wasted while waiting for the cache miss to be satisfied. Burton Smith in 1978 [3] suggested using that spare capacity to fetch from other instruction streams. The concept was first implemented on his Donelcor HEP. The concept did not show up in the microprocessor world until the 1990s, where it was expanded to allow fetching from alternate individual instruction streams in alternate cycles, but executing from all instruction streams concurrently in the same cycle, based upon the availability of the required data. The first microprocessor to implement this feature was the Pentium IV in 2000.

J. Fast Cores

On compute intensive tasks, the flow dependencies of source operands waiting for the results produced by earlier instructions can be a significant bottleneck. A solution—run the execution core at a frequency much faster than the rest of the microprocessor. The Pentium IV chip, introduced in 2000, has an operating frequency of 1.7 GHz, but an ALU that operates at 3.4 GHz.

IV. The One-Billion-Transistor-Chip Future

As we have said, within the current decade, process technology is promising one billion transistors on a single die, operating at a frequency of from 6 to 10 GHz. What will we do with all that capability?

Computer architects today do not agree on the answer. Some argue for extending the CMP idea that we described above. The argument is that with one billion transistors, we could put 100 microprocessors on a single chip, consisting of 10 million transistors each. The argument further states that a 10 million transistor processor is still very substantial, and building anything larger than that would just incur greater diminishing returns.

Others suggest an expanded use of simultaneous multithreading. They argue that many of the resources required for a CMP could be shared on a single processor SMT chip, freeing up the saved resources for other functionality such as larger caches, better branch predictors, more functional units, etc.

Some, including this author, note that while SMT is certainly an improvement over CMP with respect to shared resources, they both fall woefully short with respect to speeding up most of the important nonscientific benchmarks. The reason: Most of these benchmarks have the disappointing characteristic, re: SMT, of consisting of a single instruction stream. That and the notion that a very expensive chip ought to address problems not solvable by a multicomputer network made up of lots of smaller cheaper chips argue for using all billion transistors to produce a very high-powered uniprocessor.

Still others complain that since CAD tools are already unequal to the task of accurately validating our current chips, it is irresponsible to design even more complex ones. They

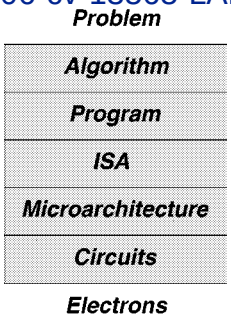


Fig. 2. The microprocessor tomorrow.

advocate a very simple core processor, combined with enormous on-chip caches.

Those that are cost-centered (a very different design point) recognize that higher levels of integration produce cheaper products, and suggest using the one billion transistors that will be available to put the entire nonaggressive system on the one chip.

These are the alternative suggestions, and I suspect each of them will show up in some product during the next ten years. My own preference is to use the billion transistors to evolve the highest performance uniprocessor that can process single-instruction-stream applications. Agents to catalyze this evolution remain as before: new requirements, bottlenecks and good fortune. Some ideas we are likely to see evolve are the following.

A. The New Microprocessor

Up to now, the microprocessor has been treated as shown in Fig. 1. But why so? If we take our levels of transformation and include the algorithm and language in the microprocessor, the microprocessor then becomes the thing that uses device technology to solve problems. See Fig. 2. Why is it not reasonable for someone wishing to design a microprocessor that addresses some point in the application space to take into account the special purpose algorithm needed to solve that problem, and embed that algorithm in the microprocessor? We do that today in low-cost embedded applications. Why not in high-performance requirements where we are willing to tolerate high costs?

This tailoring may take the form of reconfigurable logic, discussed below, special dedicated instructions in the ISA, or an integrated functional unit (like a DSP engine) provided on the chip.

B. A New Data Path

On-chip frequencies are expected to be so high that serious consideration must be given to the wire length of any signal on the chip. Some signals will require multiple cycles to traverse the chip, and one must examine carefully which signals will be allowed to do that. Most signals will probably not be allowed to. Therein lies the challenge: to redesign the data path in light of the new constraint of wire length.

C. Internal Fault-Tolerance

Another consequence of the increasing on-chip frequencies will be the susceptibility to soft errors—errors that will be created intermittently and infrequently due to the physical nature of the materials operating at the expected clock frequencies. Future microprocessors will have to provide functionality to check for and correct these soft errors as they occur.

D. Asynchronous and Synchronous Units Coexisting

Already, clock skew is a serious problem. At 6 GHz it is much worse. Admittedly, asynchronous structures are tougher to design, but they do get around the problem of a global clock that everything synchronizes on. And that is sufficiently important that the challenge is worth addressing. My expectation is that we will see structures that operate asynchronously for some fixed period of time (measured in clock cycles), after which they synchronize with the global clock. Different structures will require different amounts of time in which they need to operate asynchronously to get around their unique skew problems.

E. Different Cycle Times for Different Functions

For those structures that operate synchronously, it is not necessary that they all run at the rated frequency of the chip. Parts that don't need to go fast could be designed to go slow and save on power, for example. The future transistor budget can provide enormous flexibility to properly take advantage of the variability of the on-chip needs.

An ALU running at twice the frequency of the rest of the core is just the tip of the iceberg. The future microprocessor could use the clock intelligently, greater speed where needed, less speed where not needed, and very slow where speed is not in the critical path at all.

F. New Materials

I have no clue where these materials will come from, but Moore's law continues to prevail despite the doomsayers that show up every five years or so to spell its demise. Critical materials are needed vis-a-vis on-chip conductivity, and even more importantly, vis-a-vis power requirements and heat dissipation. So, in the spirit of unadulterated wishful thinking, I wish for engineering ingenuity to prevail again.

G. Expanded Use of Microcode

Off-chip bandwidth is expensive, on-chip bandwidth is plentiful. My expectation: we will more effectively harness on-chip bandwidth. The expanded use of microcode is a way to do that. For example, microcoded routines could exploit the spare capacity of underutilized functional units in a subordinate role to the primary instruction stream. We have coined the term subordinate simultaneous microthreading (SSMT) to reflect its role in an SMT machine [4]. These microcoded routines could perform dynamic recompilation, compute some compound instruction, tune

the cache replacement policy, or in some other way, perform a calculation that allows the primary instruction stream to execute faster.

H. Reconfigurable Logic

Consistent with Fig. 2, I expect many future microprocessors to address the requirements of specific applications. One application could make good use of some logic function that would be useless to other applications, while another application could make good use of a different logic function that would be useless to the first application. Perhaps both applications could be handled effectively by a microprocessor that had the capability to do run-time tailoring. That is, I think an on-chip structure, perhaps a low granularity FPGA, but more likely a higher granularity reconfigurable logic structure, will be common to future microprocessors.

I. Potpourri

Finally, I offer a list of features I expect to see in the high-performance microprocessor of 2008 or 2009, or whenever it is that process technology finally provides us with one billion transistors on a single silicon die.

1) Expanded use of the trace cache, where dynamic instruction stream segments will consist of far more than 8 instructions per entry, probably pre-scheduled with the help of the compiler (akin to the Block-structured ISA or the rePLay mechanism), but tuned at run-time.
2) On-chip microcode for using the spare capacity of the execution core to tune the on-chip hardware structures.
3) Dynamic recompilation of the executing program, probably carried out by the fill unit, or on-chip microcode will be commonplace.
4) Multiple (at least three) levels of cache with corresponding ISA additions (multiple prefetch and post-store instructions) to move data closer to the core and further way from the core in response to the core's need for that data.
5) Aggressive value prediction hardware, probably with a procedure level granularity, and corresponding compiler optimizations to aid its effectiveness.
6) Performance monitoring hardware to allow tuning the hardware at run-time to more effectively match the needs of the executing program.
7) An on-chip structure for monitoring and affecting the energy usage of the chip.

V. Conclusion

The microprocessor has enjoyed an exciting journey since its invention in 1971. Few technologies can boast the enormous strides it has made. Unfortunately, there are those who would argue that the end of this golden era is just around the corner. But such naysayers have been here before. They said the MIPS R2000 was all the microprocessor anyone would ever need in 1986, and ten years later they said the Intel Pentium Pro was all the microprocessor that anyone would ever need. The industry continues to do better, and the users of that technology continue to make use of that "better."

This is not to say that things will not change, that new ingenuity is no longer needed. Downstream, we may need a radical paradigm shift such as quantum computing to bail us out, but we are hardly constrained right now.

Sure we need to develop better CAD tools. Current CAD tools have trouble verifying the microprocessors of today, let alone the suggestions in this article. And, sure we need to think more broadly in our concept of the microprocessor (Fig. 2, for example). But the bottom line is that Moore's Law is alive and well, and still providing plenty of opportunity.

Acknowledgment

This introduction has benefited from many interactions over many years with many former and current students and many former and current colleagues. The draft published here has benefited explicitly from comments and criticisms of S. J. Patel, R. Belgard, and R. Ronen. It should be noted that an excellent and more detailed treatment of many of the issues touched on here are provided in a paper by Ronen and his colleagues at Intel [5], which this author recommends to the reader.

References

[1] H. Mazor, "The history of the microcomputer-invention and evolution," *Proc. IEEE*, vol. 83, pp. 1601–1608, Dec. 1995.
[2] Intel web site [Online]. Available: http://www.intel.com/pressroom/kits/quickrefyr.htm#1971.
[3] B. Smith, "A pipelined, shared resource MIMD computer," in *Proc. 1978 Int. Conf. Parallel Processing*, Aug. 1978, pp. 6–8.
[4] R. S. Chappell, J. Stark, S. P. Kim, S. K. Reinhardt, and Y. N. Patt, "Simultaneous subordinate microthreading (SSMT)," in *Proc. 26th Annu. Int. Symp. Computer Architecture*, May 1999, pp. 186–195.
[5] R. Ronen, A. Mendelson, K. Lai, S.-L. Lu, F. Pollack, and J. P. Shen, "Coming challenges in microarchitecture and architecture," *Proc. IEEE*, vol. 89, pp. 325–340, Mar. 2001.

Yale Patt (Fellow, IEEE) received the B.S. degree from Northeastern University and the M.S. and Ph.D. degrees from Stanford University, all in electrical engineering.

He is Professor of Electrical and Computer Engineering and the Ernest Cockrell, Jr. Centennial Chair at The University of Texas at Austin. He directs the Ph.D. research of nine students on problems relating to the implementation of high-performance microprocessors. He has been an active consultant to the microprocessor industry for more than 30 years. His particular love is teaching—both the first required computing course for freshmen and the advanced graduate courses in microarchitecture. He recently co-authored with S. J. Patel a textbook, *Introduction to Computing Systems: From Bits and Gates to C and Beyond* (New York: McGraw-Hill, 2000) which is a major departure from the traditional freshman course. It has already been adopted by more than 50 colleges and universities.

Dr. Patt has been awarded the IEEE/ACM Eckert Mauchly Award (1996), the IEEE Emmanuel R. Piore medal (1995), the IEEE Wallace W. McDowell medal (1999), and the ACM Karl V. Karlstrom Outstanding Educator award (2000). He is a Fellow of the ACM.

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

SERIAL NUMBER	FILING DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NO.
08/414,250	03/31/95	SCHWARZ E	P0994-050

24M1/0411

CHRISTOPHER A HUGHES
MORGAN AND FINNEGAN
345 PARK AVENUE
NEW YORK NY 10154

EXAMINER	
NGO, C	

ART UNIT	PAPER NUMBER
2306	8

DATE MAILED: 04/11/97

NOTICE OF ALLOWABILITY

PART I.

1. ☑ This communication is responsive to papers filed on 12/31/96
2. ☑ All the claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice Of Allowance And Issue Fee Due or other appropriate communication will be sent in due course.
3. ☑ The allowed claims are 1-17
4. ☐ The drawings filed on _____ are acceptable.
5. ☐ Acknowledgment is made of the claim for priority under 35 U.S.C. 119. The certified copy has [_] been received. [_] not been received. [_] been filed in parent application Serial No. _____ filed on _____
6. ☐ Note the attached Examiner's Amendment.
7. ☐ Note the attached Examiner Interview Summary Record, PTOL-413.
8. ☑ Note the attached Examiner's Statement of Reasons for Allowance.
9. ☐ Note the attached NOTICE OF REFERENCES CITED, PTO-892.
10. ☐ Note the attached INFORMATION DISCLOSURE CITATION, PTO-1449.

PART II.

A SHORTENED STATUTORY PERIOD FOR RESPONSE to comply with the requirements noted below is set to EXPIRE THREE MONTHS FROM THE "DATE MAILED" indicated on this form. Failure to timely comply will result in the ABANDONMENT of this application. Extensions of time may be obtained under the provisions of 37 CFR 1.136(a).

1. ☐ Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL APPLICATION, PTO-152, which discloses that the oath or declaration is deficient. A SUBSTITUTE OATH OR DECLARATION IS REQUIRED.
2. ☑ APPLICANT MUST MAKE THE DRAWING CHANGES INDICATED BELOW IN THE MANNER SET FORTH ON THE REVERSE SIDE OF THIS PAPER.
   a. ☑ Drawing informalities are indicated on the NOTICE RE PATENT DRAWINGS, PTO-948, attached hereto or to Paper No. 5. CORRECTION IS REQUIRED.
   b. ☐ The proposed drawing correction filed on _____ has been approved by the examiner. CORRECTION IS REQUIRED.
   c. ☐ Approved drawing corrections are described by the examiner in the attached EXAMINER'S AMENDMENT. CORRECTION IS REQUIRED.
   d. ☑ Formal drawings are now REQUIRED.

Any response to this letter should include in the upper right hand corner, the following information from the NOTICE OF ALLOWANCE AND ISSUE FEE DUE: ISSUE BATCH NUMBER, DATE OF THE NOTICE OF ALLOWANCE, AND SERIAL NUMBER.

Attachments:
- Examiner's Amendment
- Examiner Interview Summary Record, PTOL-413
- ☑ Reasons for Allowance
- Notice of References Cited, PTO-892
- Information Disclosure Citation, PTO-1449
- Notice of Informal Application, PTO-152
- Notice re Patent Drawings, PTO-948
- Listing of Bonded Draftsmen
- Other

CHUONG D. NGO
PATENT EXAMINER
GROUP 2300

CASE 19623185 ATTY TJR
DUE DATE July 11, 1997
STATUTORY DATE October 11, 1997
PTOL-37 (REV. 4-89) BY KB

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE

Attachment to Paper Number
Serial No.

INFORMATION ON HOW TO EFFECT DRAWING CHANGES

1. **Correction of Informalities**—37 CFR 1.85; 1097 OG 36

IN APPLICATIONS FILED **BEFORE** JANUARY 1, 1989 OPTION a) OR b) MAY BE USED IN ORDER TO CORRECT ANY INFORMALITY IN THE DRAWING.

IN APPLICATIONS FILED **AFTER** JANUARY 1, 1989 ONLY OPTION a) **MAY BE USED.**

AFTER JANUARY 1, 1991 ONLY OPTION a) MAY BE USED REGARDLESS OF FILING DATE.

a) File new drawings with the changes incorporated therein. The art unit number, serial number and number of drawing sheets should be written on the reverse side of the drawings. Applicant may delay filing of the new drawings until receipt of the "Notice of Allowability" (PTOL-37). If delayed, the new drawings **MUST** be filed within the **THREE MONTH** shortened statutory period set for response in the "Notice of Allowability" (PTOL-37). Extensions of time may be obtained under the provisions of 37 CFR 1.135(a). The drawing should be filed as a separate paper with a transmittal letter addressed to the Official Draftsman.

b) Request a commercial bonded drafting firm to make the necessary corrections. A bonded draftsman must be authorized, the corrections executed and the corrected drawings returned to the office during the **THREE MONTH** shortened statutory period set for response in the "Notice of Allowability" (PTOL-37). Extensions of time may be obtained under Provisions of 37 CFR 1.136(a).

Timing of Corrections

Applicant is required to submit **acceptable** corrected drawings within the three month shortened statutory period set in the "Notice of Allowability" (PTOL-37). Within that three month period, two weeks should be allowed for review by the Office of the correction. If a correction is determined to be unacceptable by the Office, applicant must arrange to have acceptable correction re-submitted within the original three month period to avoid the necessity of obtaining an extension of time and paying the extension fee. Therefore, applicant should file corrected drawings as soon as possible.

Failure to take corrective action within set (or extended) period will result in **ABANDONMENT** of the Application.

2. **Corrections other than Informalities Noted by the Draftsman on the PTO-948**

All changes to the drawings, other than informalities noted by the Draftsman, **MUST** be made in the same manner as above except that, normally, a red ink sketch of the changes to be incorporated into the new drawings **MUST** be approved by the examiner before the application will be allowed. No changes will be permitted to be made, other than correction of informalities, unless the examiner has approved the proposed changes.

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
ASSISTANT SECRETARY AND COMMISSIONER
OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

Dear Patent Office Customer:

The Technical Support Staff of Group 2400 has undertaken continuous quality improvement efforts to ensure that the accompanying correspondence meets high quality standards and focuses on good customer service. It is important to us that you are satisfied with the services we provide.

If the contents of the correspondence have any clerical omissions (such as missing references or pages), illegible text, or other problems or concerns of this nature which you wish to bring to my attention, please call or fax me as soon as possible. I will take the appropriate action to expedite the necessary corrections.

Doretha A. Marcelli
Group 2400, Chief SLIE

Phone No. (703) 305-4762
Fax No. (703) 308-6743

REASONS FOR ALLOWANCE

1. The following is an Examiner's Statement of Reasons for Allowance:

The prior art of record does not fairly suggest in a computer system a conversion means as recited in claims 1, and the first through forth converter as recited in claim 11 wherein, in accordance with 35 USC 112, 6th paragraph, the conversion means is construed to cover the corresponding structure of the combination of converters 11,12,15,16,22,24, multiplexers 28,28,29 and register 13,14,14 connected as disclosed in figure 1 or equivalents thereof, and the first and forth converter apply the same transformation for both converting an operand of a first floating point architecture type to the internal floating point format, and transforming an operand of a second floating point architecture type as recited in claim 11.

2. Any comments considered necessary by applicant must be submitted no later than the payment of the Issue Fee and, to avoid processing delays, should preferably accompany the Issue Fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."

3. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Chuong D. Ngo whose telephone number is (703) 305-9764.

Any inquiry of a general nature or relating to the status of this application should be directed to the Group receptionist whose telephone number is (703) 305-3800

The fax number for this Group is (703) 305-9724.

Chuong D. Ngo
Primary Examiner
Art Unit 2306

04007-97